VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

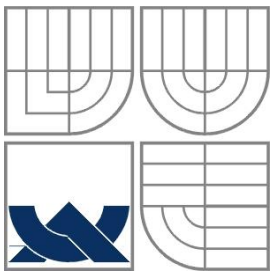
ZÁVODNÍ HRA VE 3D

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

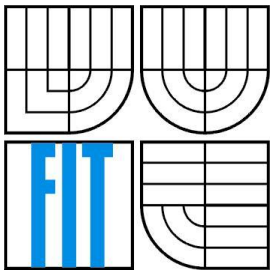
AUTOR PRÁCE
AUTHOR

MARTIN ŠEVČÍK

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ZÁVODNÍ HRA VE 3D

RACING GAME IN 3D

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN ŠEVČÍK

VEDOUCÍ PRÁCE
SUPERVISOR

ŠUŠKA BORIS, ING.

BRNO 2010

Abstrakt

Táto bakalářská práce se zabývá problematikou řešení interakce Java Monkey Engine (jME) a modelů vytvořených v programu Autodesk 3D Studio Max. Obsahuje teoreticky základ k architektuře jMonkey Engine a herní knižnici Lightweight Java Game Library (LWJGL). Dále popisuje techniky při práci s objekty ve 3D scéně a jejich způsob implementace.

Abstract

This bachelor's thesis describes issues of interaction between Java Monkey Engine (jME) and models created with program Autodesk 3D Studio Max. It contains theoretical basis for the jMonkey Engine architecture and Lightweight Java Game Library (LWJGL). In the following section, the thesis describes techniques of working with objects in the 3D scene and their way of implementation.

Klíčová slova

Java Monkey Engine, jME, jMonkey Engine, Lightweight Java Game Library, LWJGL, 3D scéna, závodní hra ve 3D, 3D hra, 3D modely, 3D Studio Max, 3DS Max.

Keywords

Java Monkey Engine, jME, jMonkey Engine, Lightweight Java Game Library, LWJGL, 3D scene, racing game in 3D, 3D game, 3D models, 3D Studio Max, 3DS Max.

Citace

Ševčík Martin: Závodní hra ve 3D, bakalářská práce, Brno, FIT VUT v Brně, 2010

Závodní hra ve 3D

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Borisa Šušku
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Ševčík
6. dubna 2010

Poděkování

Chcel by som poďakovať môjmu vedúcemu Ing. Borisovi Šuškovi za odborné vedenie, poskytnuté konzultácie a za ústretovosť pri výbere zadania. Rovnako sa chcem poďakovať mojej rodine a priateľom za ich podporu pri vytváraní tejto bakalárskej práce.

Obsah

1 Úvod

Modelovanie 3D objektov je v dnešnej dobe jedným z veľmi rýchlo sa rozvíjajúcich odvetví počítačovej grafiky. Hlavnou príčinou prečo je tomu tak je, že nároky užívateľov z pohľadu vizuálnej stránky sa neustále zvyšujú. Aj to je dôvod, prečo sa čoraz častejšie stretávame s 3D modelmi, či už v hernom alebo filmovom priemysle. Okrem výsledného vzhľadu týchto priestorových modelov, sú pri vytváraní počítačových hier kladené požiadavky na rýchlosť výpočtu a súčasne na čo najmenšie zaťaženie procesoru.

Pri tvorbe počítačovej hry väčšina programátorov uprednostní možnosť použitia 3D modelov vytvorených v externých modelovacích programoch, ako napríklad 3D Studio Max, Maya, Blender, pred pracným a zdĺhavým vytváraním týchto zložitých modelov len pomocou jednoduchých základných útvarov implementovaných v danom programovacom jazyku, u ktorých je potrebné dohliadať na ich výslednú harmóniu. Tým je možné dosiahnuť vytvorenie dokonalých a po vizuálnej stránke prepracovaných 3D modelov.

Táto práca prináša demonštráciu prepojenia externých 3D modelov s hernou knižnicou Java Monkey Engine (jME) a to na praktickom príklade vytvorenia trojrozmernej počítačovej hry v prostredí Java SE z 3D modelov vymodelovaných v programe 3D Studio Max.

Úvodné kapitoly č. 2 a č.3 popisujú jemne úvod do prostredia Java Monkey Engine, jeho históriu, architektúru, objekt kamery, či graf scény. Kapitola č.4 sa stručne venuje architektúre hernej knižnice Lightweight Java Game Library a jej výhodám pri vytváraní počítačových hier.
Kapitola č. 5 pojednáva o práci s 3D modelmi v jME. V krátkosti popisuje možné formáty týchto modelov. Kapitola č.6 stručne pojednáva o histórii programu Autodesk 3D Studio Max, spôsobe vytvorenia polygónu a transformáciách objektov v jeho prostredí.

Záverečné kapitoly č. 7 a č. 8 sa venujú výberu herného typu pre počítačovú hru, jeho štruktúre a následne podrobne popisujú implementáciu počítačovej hry v prostredí Java Monkey Engine a použité techniky pri práci s 3D modelmi.

Výsledkom tejto práce je vytvorená počítačová hra v prostredí 3D za použitia externých 3D modelov, ktorá sa snaží demonštrovať toto prepojenie v atraktívnom podaní.

2 Java Monkey Engine

Táto kapitola sa venuje v krátkosti histórii Java Monkey Engine (jME, jMonkey Engine) a ďalej popisuje prehľad architektúry jMonkey Engine a objektu kamery. Kapitola bola čerpaná z [2].

2.1 História

Java Monkey Engine bol vytvorený roku 2003 autorom menom Mark Powell. Na konci toho istého roka sa k tímu pripojil Joshua Slack a stal sa jeho neoddeliteľnou súčasťou. Títo hlavní predstavitelia spolu s ďalšími členmi vývojárskeho tímu vyvíjali jMonkeyEngine od základnej verzie až po verziu 2.0 a podarilo sa im založiť komunitu, ktorá existuje dodnes.

Od augusta 2008 prešiel projekt vývoja jME viacerými zmenami. Počiatočný vývojový tím bol nahradený novými členmi. Novým vedúcim programátorom ako aj jediný architekt a vývojár novej verzie jME 3.0 sa stal Kirill Vainer [3].

2.2 Architektúra jME

Java Monkey Engine bol navrhnutý ako vysokorýchlostný real-time grafický engine. Je implementovaný v programovacom jazyku Java. S pokrokmi vo vývoji hardvéru, počítačových grafických kariet a taktiež s pokročilým vývojom programovacieho jazyka Java sa ukázalo, že je nevyhnutné a potrebné vytvoriť pre Javu hernú knižnicu. A jME je práve touto hernou knižnicou, ktorá spĺňa mnohé z potrieb herných vývojárov práve poskytovaním vysoko výkonného vykresľovacieho systému grafu scény. Tento grafický engine využíva najnovšie funkcie OpenGL, ktoré umožňujú naplno využiť možnosti zobrazovania a vykresľovania najmodernejších grafických kariet za predpokladu jednoduchého používania zo strany herných vývojárov.

jME sa môže pochváliť s veľmi prijateľným intuitívnym ovládaním pre užívateľa, čo zvyšuje silu tohto programovacieho jazyka. Ďalšou výhodou je, že dokázal spojiť intuitívne rozhranie s pokročilými technikami moderného grafického programovania. Jeho modulárna architektúra zaisťuje možnosť rýchleho prispôsobenia sa novým vylepšeniam z oblasti grafického hardwaru a to aj vďaka tomu, že OpenGL neustále zlepšuje svoju prispôsobivosť a zdokonaľuje svoje vykresľovacie schopnosti v súčinnosti s najnovšími grafickými kartami. Preto je jME pomerne rýchlo schopné využívať tieto nové vylepšenia a prispôsobiť sa daným zmenám vďaka svojej architektúre takmer okamžite.

Jadrom jME systému je graf scény. Graf scény je dátová štruktúra, ktorá obsluhuje dáta celého virtuálneho sveta. Vzťahy medzi dátami v hre napríklad geometria, zvuk, fyzikálne zákony a podobne sú uchovávané v stromovej štruktúre s listovými uzlami reprezentujúcimi základné prvky

hry. To sú práve elementy, ktoré sú zvyčajne vykresľované do scény alebo prehrávané prostredníctvom zvukovej karty.

Organizácia grafu scény je veľmi dôležitá a väčšinou závislá od druhu a typu aplikácie. Keďže graf scény je väčšinou reprezentovaný veľkým množstvom dát, je tu možnosť tieto dáta rozdeliť na menšie, ľahšie ovládateľné a dostupné elementy. Obyčajne sú tieto elementy zoskupované a poprepájané vzájomnými vzťahmi, veľmi často priestorovým rozmiestnením. Toto zoskupenie umožňuje odstrániť veľké sekcie dát hry, ktoré nie je potrebné vykresľovať na aktuálnej scéne. Napríklad ak nie je potrebné, aby sa zobrazovala určitá sekcia virtuálneho sveta, získame tým rýchlejšie spracovávanie dát, pretože zaberieme menej času CPU a GPU potrebného pri ich spracovaní a vďaka tomu je rýchlosť hry zlepšená.

Kým graf scény je jadro grafických prvkov v jME, aplikačné nástroje poskytujú rýchlu možnosť vytvorenia grafického obsahu a spustiť hlavnú slučku hry. Vytvorenie herného okna, vstupného systému, kamerového a herného systému obsluhuje volanie jednej, či dvoch metód. To umožňuje programátorom stráviť viac času prácou na ich vlastnej aplikácií, než nad zdĺhavým implementovaním metód potrebných k spusteniu aplikácie. Zobrazovací a vykresľovací systém umožňuje oddelene komunikovať s grafickou kartou a to z dôvodu, aby bolo možné spracovať veľké množstvo dát prostredníctvom vykresľovacieho systému, ktorý je pre užívateľa neviditeľný počas celého behu aplikácie. Hlavné slučky hry obstarávajú možnosť jednoduchého rozšírenia a zaručujú jednoduchý vývoj aplikácie.

Skutočné vykresľovanie grafu scény je skryté pred užívateľom. To má za výhodu možnosť prepínania medzi rozdielnymi vykresľovacími systémami bez prepisovania jediného riadku kódu aplikácie. Ak aplikácia pre spustenie používa knižnicu LWJGL (Lightweight Java Game Library) od Puppy Games, nie je problém prepnúť na druhú knižnicu JOGL (Java OpenGL) od spoločnosti Sun Microsystems [4]. Je možné, že sa vyskytnú nejaké menšie rozdiely vo výslednom zobrazení, ale nie je potrebná prestavba celého projektu. Avšak zameranie vývojárov sa postupne prenieslo na pridávanie nových vylepšení iba do jedného vykresľovacieho systému, čo malo za následok zastaralosť toho druhého a tým pádom aj jeho neatraktívnosť. Práve z tohto dôvodu je v súčasnosti podporovaný iba jeden vykresľovací systém a ním je práve LWJGL [5].

Každý nástroj jME systému bol postavený na základe jednoduchých stavebných blokov. Takže všetky grafické elementy dedia z triedy `Spatial`, všetky vstupy a im pridružené akcie dedia z triedy `InputAction` a podobne. Tým je zaistená konzistencia, vďaka ktorej používanie pokročilejších metód a technológií bude podstatne jednoduchšie a pomocou nástrojov obsiahnutých v jME je vytvorenie aplikácie rýchle a efektívne [6].

2.3 Objekt Kamery

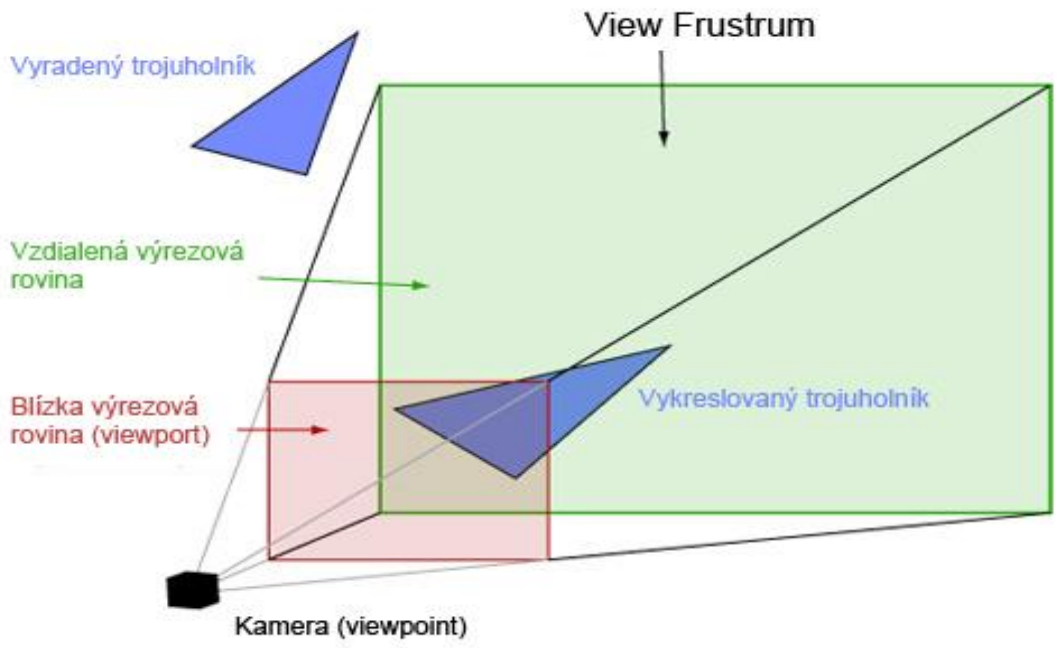
jME využíva triedu `Camera`, ktorá obsahuje niekoľko kľúčových nastavení. Tieto nastavenia majú za následok ako bude výsledná renderovaná scéna vyzerať. Teda koľko a čo bude z virtuálneho sveta vykreslené a zobrazené pre koncového užívateľa.

Kamerou môže byť zaobchádzané rovnako ako s akýmkoľvek iným objektom v scéne, ktorý môže byť pripojený do scény ako uzol alebo ako jednoduchý objekt. To si však vyžaduje použitie `CameraNode`.

2.3.1 CameraNode

`CameraNode` je uzol, do ktorého pripojíme vytvorenú kameru. To umožňuje využiť objekt kamery ako súčasť scény. Prínosom je, že je následne možné pripojiť kameru na akýkoľvek objekt v danej scéne a jej ďalšie použitie je pomerne jednoduché a zautomatizované. Preto je väčšinou zaobchádzané s kamerou ako s elementom scény. To umožní jej pripojenie v akejkoľvek výške stromu grafu scény, čo otvára veľa možností využitia. Napríklad pripojenie kamery na lietadlo, takže nie je potrebné sa obávať pohybu lietadla, pretože kamera sa bude pohybovať automaticky s ním. Preto lietadlo nikdy nezmizne zo zorného uhlu kamery a teda aj užívateľa. Taktiež je možné pripojiť uzly na kameru, čím sa dá vytvoriť pohľad z prvej osoby.

Umiestnenie a nastavenia kamery definuje View Frustrum, čo je logický popis celkového objemu priestoru, ktorý je viditeľný pre užívateľa.



Obrázok 2.1: View Frustrum

Takto novo vytvorený vrchol je v podstate obdĺžnik, ktorý reprezentuje obrazovku užívateľa. To umožňuje rozšírenie zorného poľa. Všetko v rámci pyramídy je vykresľované, zatiaľ čo ostatné objekty nie je potrebné vykresľovať. Práve vďaka tejto možností je jME schopné zvládnuť zložité scény zložené z mnohých objektov pri zachovaní vysokého počtu snímok za sekundu (FPS).

Nastavenia kamery budú teda definovať, kde je View Frustrum umiestnené, jeho orientáciu v rámci virtuálneho sveta a vlastnosti jeho bočných stien. Všetky tieto nastavenia teda dávajú možnosť vytvoriť si pohľad na svet taký, ako sám autor požaduje, či už náhľad z veľkej výšky so širokým zorným uhlom alebo tunelové videnie pri umiestnení kamery čo najnižšie. A práve tieto možnosti vyzdvihujú schopnosti jMonkey Engine.

3 Graf scény

Táto kapitola sa venuje popisu grafu scény v prostredí jME. Zdroj informácií bol čerpaný z [7].

Graf scény je hierarchické zoskupenie uzlov podľa priestorového umiestnenia. Tento strom má listové uzly, ktoré reprezentujú dáta, ktoré majú byť spracované a vnútorné uzly, ktoré tieto dáta pomáhajú spravovať.

Existujú štyri hlavné dôvody, prečo je takáto organizácia scény výhodná pri tvorbe hier :

- Dáta v hre (objekty, prostredie a ďalšie) sú zvyčajné veľké. Sú normálne usporiadané v priestore podľa ich polohy, takže tieto dáta môžu byť jednoducho zoskupené v priestore v 3D svete. Keďže svet je organizovaný týmto spôsobom, je prirodzené, že umiestnenie napríklad LightNode (uzol predstavujúci svetlo) ovplyvní ďalšie elementy pod ním. Toto je zaobstarané automaticky grafom scény.
- Takáto hierarchická štruktúra poskytuje referenčný bod pre priestorové umiestnenie jednotlivých objektov prostredníctvom koreňového uzla. Koreňový uzol obsahuje všetky objekty k nemu pripojené. Vďaka tomu je možné jednoducho odstrániť irelevantné časti grafu scény.
- Veľa elementov vo virtuálnom svete je reprezentovaných takouto stromovou štruktúrou, a preto je ľahké zobraziť ich štruktúru. Pozícia a rotácie uzlov je dedená smerom nadol. To znamená, že ak nadradený uzol zaznamená pohyb alebo rotáciu, tak aj jeho podriadené uzly dedia danú operáciu a tá sa prejaví aj na ich polohe.
- Je nenáročné vyjadriť graf scény v iných nástrojoch. To umožňuje scéne, aby bola vytvorená v editore a napríklad prostredníctvom XML ju jednoducho importovať do jME.

Graf scény je tvorený dvoma typmi uzlov a to vnútornými a listovými. Vnútorné uzly sa nazývajú uzlami a listové uzly sa nazývajú geometrie. Uzly môžu obsahovať potomkov (iný uzol alebo geometriu), zatiaľ čo geometria potomkov obsahovať nemôže. Každý uzol ako vnútorný tak aj listový obsahuje dôležité informácie o sebe samom : transformácia, ohraničujúci zväzok, vykresľovací stav a kontroléry.

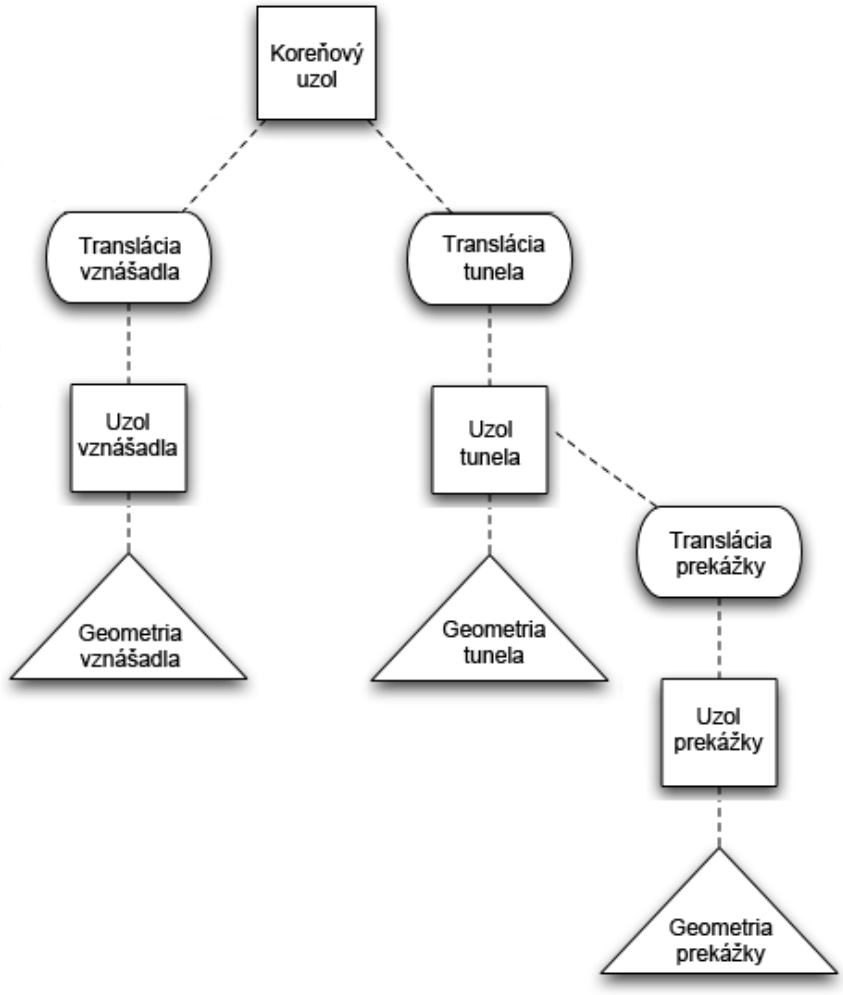
Transformácia (Transformation) definuje orientáciu, polohu a veľkosť uzlu a vzťahuje sa aj na potomkov, takže rotácia rodiča ovplyvní aj všetkých jeho potomkov.

Ohraničujúci zväzok (Bounding Volume) definuje oblasť, ktorá minimálne obsahuje veľkosť príslušného uzlu a jeho vrcholov. Preto zväzok na úrovni geometrie obsahuje všetky jeho vrcholy. Avšak zväzok na úrovni uzla pokrýva oblasť aj všetkých jeho potomkov.

Stav vykresľovania (Render State) definuje ako budú jednotlivé elementy vykresľované do scény. To ďalej ovplyvní aj ich potomkov. Toto zaručuje minimálne prepínanie stavu, teda ak uzol obsahuje tisíce rovnakých geometrií, tento uzol môže obsahovať jeden vykresľovací stav pre textúru, ktorý nastaví rovnakú textúru pre všetkých tisíc geometrií.

Kontroléry (Controllers) slúžia na definovanie modifikácii uzlov za časovú jednotku. To zvyčajne zahŕňa animácie modelu, fyzikálne vlastnosti (hodenie loptičkou) a podobne. Ovládače samé o sebe nemôžu ovplyvniť svojich potomkov avšak môžu mať efekt na uzol.

Tento príklad ukazuje možné navrhnutie organizácie hry vznášadla v tuneli. Existujú tri hlavné typy geometrie: tunel, vznášadlo a prekážka. Tunelu by malo byť umožnené byť voľne umiestnený v priestore, a preto je pripojený ku koreňovému uzlu. Pre zachovanie voľného pohybu vznášadla v priestore je vznášadlo taktiež ako tunel pripojené ku koreňovému uzlu. Prekážka sa môže nachádzať iba vo vymedzenom priestore, a to práve vo vnútri tunela. A z dôvodu, aby sa pri zmene polohy tunela zaručil zároveň aj pohyb prekážky, je uzol pripojený k uzlu tunela.



Obrázok 3.1 : Graf scény- príklad hry vznášadla v tuneli

4 Lightweight Java Game Library

Táto kapitola stručne popisuje architektúru knižnice Lightweight Java Game Library (LWJGL). Informácie sú čerpané z [5].

LWJGL je riešením pre programovanie kvalitných hier v jazyku Java ako pre profesionálnych, tak aj pre začínajúcich Java programátorov. LWJGL poskytuje vývojárom prístup k vysoko-výkonným medzi-platformovým knižniciam ako napríklad **OpenGL** (Open Graphics Library) a **OpenAL** (Open Audio Library), ktoré umožňujú tvorbu najmodernejších 3D hier s priestorovým 3D zvukom. Navyše LWJGL dokáže zariadiť prístup k ovládačom ako gamepad, volant, či joystick, čím sa ovládanie hry môže stať zábavnejšie, jednoduchšie a vierohodnejšie.
LWJGL nebol vytvorený za účelom programovať hry jednoducho, ale primárne za účelom sprístupniť technológiu, ktorá umožní vývojárom dosiahnuť zdroje, ktoré sú inak nedostupné alebo len veľmi slabo implementovateľné na existujúcej platforme Javy. Predpokladá sa, že prostredníctvom dlhodobejšieho vývoja a možnosťami rozšírenia bude knižnica LWJGL základom pre programovanie komplexnejších herných knižníc a herných enginov. LWJGL je dostupná pod BSD licenciou, čo znamená že je open source a jeho používanie nie je nijako spoplatnené.

Pomocou LWJGL je možné dosiahnuť :

- Rýchlosť
- Jednoduchosť
- Všeobecnosť
- Šetrnosť
- Bezpečnosť
- Priamosť
- Minimalizmus

Rýchlosť

Hlavným cieľom LWJGL bolo zvýšiť rýchlosť vykresľovania Javy. Preto sa napríklad vyhodili metódy určené pre efektívne programovanie v C, ktoré nemajú zmysel pre Javu, napríklad metóda `glColor3fv`.

Knižnica umožňuje hlásiť výnimku, ak na danom prístroji nie je dostupná hardware-ová akcelerácia, pretože nemá zmysel spúšťať aplikáciu s nízkym fps.

Jednoduchosť

LWJGL potrebuje byť jednoduchá, pretože ju využíva široké spektrum vývojárov. Preto na jednej strane nie je ťažké sa rýchlo a efektívne oboznámiť s možnosťami tejto knižnice a použiť to v praxi aj pre začiatočníkov a na druhej strane ponúka profesionálnym vývojárom veľké možnosti ako svoje skúsenosti premietnuť do tvorby aplikácie. Autori v nej pracujú s dvoma paradigmami.
Prvým, ktorý skutočne po všetkých stránkach vyhovuje pre OpenGL a druhým, ktorý zasahuje presne cielené platformy obsiahnutých od najmenších prístrojov ako napríklad PDA až po desktopové riešenia. Preto z hernej knižnice LWJGL je odstránená veľká škála nepotrebných konštrukcií, ktoré herní programátori vôbec nevyužívajú a nepotrebujú k vývoju aplikácií. A pretože konzistencia je lepšia ako komplexnosť je v knižnici namiesto pomalých polí používaný zásobník, vďaka čomu bola z knižnice vypustená možnosť volania jednej a tej istej metódy viacerými spôsobmi, a tým predchádzať zahlcovaniu knižnice.

Všeobecnosť

Knižnica LWJGL je navrhnutá, aby mohla spoľahlivo fungovať na veľkom množstve zariadení od malých ako napríklad telefóny až po multiprocesorové vykresľovacie servery. Síce v súčasnosti žiadne telefóny alebo konzoly nemajú dostatočne rýchle JVMs (Java Virtual Machine) [8] a 3D akceleráciu, avšak LWJGL je vyvíjaná s ohľadom na tieto budúce možnosti. Aj preto je v nej implementovaná podpora **OpenGL ES** (OpenGL Embeded System). Čo je štandard pre akcelerované 3D grafické vstavané systémy. To dosiahla aj vďaka tomu, že jej binárna distribúcia má menej ako 1MB a pritom dokáže zaobstarať 3D zvuk, grafiku, vstupy a výstupy. To určite ocenia menšie zariadenia, ktoré nemajú veľký pamäťový priestor.

Šetrnosť

Malé a šetrné sa rovná:

- Jednoduché. Čím je menej spôsobov ako niečo spraviť, tým je to jednoduchšie sa naučiť.
- Nechybové. Zdrojový kód obsahuje minimálne množstvo chýb.
- Stiahnuteľné. LWJGL zaberá dostatočne málo priestoru, čo mu umožňuje byť stiahnuteľný každou aplikáciou, ktorá ho dokáže používať.
- J2ME. Java 2 Micro Edition je súčasťou programovacieho jazyka Java spoločnosti Sun Microsystems. Táto platforma je určená k programovaniu spotrebnej elektroniky počnúc mobilnými telefónmi, rôznymi PDA až po špecifické moduly [9].

Bezpečnosť

Vývojári do tejto knižnice zaviedli zvýšenú bezpečnosť.

- Začal sa používať zásobník (buffer).
- Postupne sú pridávané ďalšie kontroly, ktoré majú za účel zabezpečiť, aby nedošlo k pretečeniu zásobníka, a tým zaručiť zabránenie nežiaducich útokov cez zásobník.

Robustnosť

Keďže spoľahlivý systém je oveľa užitočnejší a cennejší ako systém, ktorý dbá iba na rýchlosť, boli aplikácie vytvorené pomocou LWJGL podrobené mnohým testom pre získanie skutočných dát prostredníctvom benchmarkov [5]. A na základe týchto testov výkonnosti boli z knižnice odstránené assert makrá a nahradené oveľa jednoduchšími podmienkami **if{}** alebo **try{} catch{}** sekciami. Navyše boli všetky kontroly GL chýb presunuté z natívneho kódu do Java kódu, čím sa zaistilo, že nie je naďalej potrebná oddelená DLL knižnica pre potrebu debugovacieho módu.

Minimalizmus

To je ďalší kľúčový faktor, ktorý hral úlohu v navrhovaní LWJGL. To čo nie je potrebné, aby bolo v knižnici, tak to tam jednoducho nie je. Pôvodným zámerom pri vyvíjaní tejto knižnice bolo vytvoriť knižnicu, ktorá bude poskytovať požadované minimum potrebné pre prístup k hardwaru , ku ktorému Java prístup nemala. A v takomto duchu sa aj naďalej vedie vývoj tejto knižnice. Aj preto bol napríklad odstránený GLU (OpenGl Utility Library) [4], pretože bol väčšinou nepotrebný pre herných vývojárov.

5 Práca s modelmi v jME

Táto kapitola pojednáva o práci s 3D modelmi v jME. V krátkosti popisuje možné formáty súborov, ktoré môžu byť použité. Zdroj informácií bol čerpaný z [10].

Slovom model je v jME myslená perzistentná a obnoviteľná reprezentácia objektu v 3D virtuálnom svete. Veľkým prínosom toho, že nie je potrebné daný model pracne vytvárať v jME rôznymi základnými útvarmi a dohliadať na ich výslednú harmóniu, ale stačí jednoducho vytvoriť model v externom 3D grafickom editore je ten, že takýmto spôsobom je možné dosiahnuť kvalitné grafické efekty a taktiež vytvoriť dokonalé prepracované modely, či už pre aplikáciu alebo hru, čo určite vyzdvihuje a uľahčuje prácu pri vývoji týchto zložitých programov. Prínos je naozaj obrovský, pretože je pomerne jednoduché si vytvoriť objekty pre aplikáciu v externom programe bez zdĺhavého programovania v Jave ako napríklad v programoch Maya, Blender, 3D Studio Max. jME dokáže načítať široké spektrum najznámejších formátov :

- 3DS – formát programu Autodesk 3D Studio Max
- Ase – ASCII Scene Exporter, jedná sa o textovú podobu 3DsMax formátu [11]
- Md2 – modely z hry Quake2
- Md3 – modely z hry Quake3
- Md5 – modely z hry Doom3
- Ms3d – MilkShape modely
- Obj – Wavefront 3D objekty
- X3d – ISO štandard XML súborový formát pre reprezentáciu 3D grafiky

Pre načítanie týchto modelov do prostredia jME slúžia samostatné triedy `XxxToJme`, kde Xxx je označenie formátu, ktoré dokážu tieto modely prekonvertovať do formátu, ktorý využíva jME. Tieto triedy sú obsiahnuté v knižnici jME v balíku `com.jmex.model.converters`, a preto nie je potrebné ich zdĺhavé vytváranie. Niektoré z týchto modelov ako napríklad md5, ktorý je jeden z najviac pokročilých 3D formátov, plne podporujú kostrovú animáciu a textúrovanie, čo určite dodá hodnotu modelu na jeho prepracovanosti a na dobrom umeleckom dojme.

Tieto formáty môžu zvyčajne obsahovať viacero vlastností. Ukladajú v sebe napríklad 3D informácie ohľadom nastavenia kamery, svetla, textúry alebo systémového nastavenia programu.

Taktiež je dobré spomenúť formát **COLLADA** (COLLAborative Design Activity). Tento modelový formát bol vytvorený skupinou Khronos Group. Je to otvorený štandard, ktorý sa neustále vyvíja. Tento exportér modelov je k dispozícii pre modely z programov 3DStudio Max, Maya a Blender. Modely sú importované do jME prostredníctvom triedy `ColladaImporter`, ktorá vytvorí vhodný graf scény z elementov obsiahnutých v danom súbore. Importér v súčasnosti taktiež podporuje geometriu a kostrovú animáciu [12].

6 Autodesk 3D Studio Max

Táto kapitola sa venuje v krátkosti histórii programu Autodesk 3D Studio Max a popisuje základný prehľad jeho architektúry. Kapitola bola čerpaná z [17].

6.1 História

Autodesk 3D Studio Max (3DS Max) debutoval na trhu v roku 1990 ako vizualizačný nástroj pre profesionálov v architektúre, stavebníctve a inžinierstve hľadajúcich spôsob ako previesť ich 2D CAD/CAM výkresy do 3D prostredia, ktoré by ich nápady dokázal zobraziť zo všetkých uhlov. Každá animácia bola závislá od svetiel, prípadne kamery, ktorými sa dalo pohybovať. Napriek tomu však tento produkt dokázal vykresliť obrázky, ktorých kvalita presahovala možnosti všetkých dovtedy existujúcich programov. Neskôr sa v tomto programe začali zohľadňovať aspekty pre tvorbu 3D grafiky pre interaktívne počítačové hry a multimédia. A tak 3DS Max, predstavený v roku 1996, zaznamenal výrazný posun od iba vizualizačného nástroja s možnosťou obmedzenej animácie až k prostrediu s podporou plnej animácie.

3DS Max sa skladá z veľkého počtu rozšírení organizovaných okolo hlavného jadra programu. A to vďaka integrovanému vývojárskemu prostrediu, ktoré umožňuje skúseným programátorom v jazyku C tieto rozšírenia doplňovať a taktiež vďaka vlastnému skriptovaciemu jazyku Maxscript, v ktorom je taktiež možné vytvárať rôzne rozšírenia aj pre menej skúsených užívateľov. Zásluhou týchto možností sa 3DS Max uplatnilo vo filme, počítačových hrách, simuláciách, vizualizáciách a 3D grafike na internete.

6.2 Transformácie

3DS Max je v podstate polygonálny modelovací nástroj. Práca s polygónmi je veľmi efektívna, pretože pre lokalizáciu bodu v priestore sú potrebné iba tri súradnice (X, Y, Z). Pre vytvorenie čiary je potrebné vytvoriť dva body a funkciu, ktorá tieto dva body spojí. Pre vytvorenie plochy je potrebné lokalizovať tri body, spojiť ich v trojuholník a vymaľovať jednu stranu. Po takomto spojení vzniká plocha v priestore zvaná polygón.

V programe 3DS Max sa všetky transformácie objektov (pohybovanie, rotovanie, škálovanie) odohrávajú v prostredí zvanom Svet vesmíru. Vymodelovanie objektu prebieha pomocou manipulácie jeho geometrických komponentov (sub-objektov), napríklad vrchol, hrana, plocha a polygón, použitím ich vlastného súradnicového systému nazvaného Lokálne súradnice. Transformácie týchto komponentov na sub-objektovej úrovni sú zvyčajne brané ako sub-objektové modelovanie.

7 Výber herného typu

Táto kapitola popisuje v krátkosti výber typu hry z ponuky jME a jej hlavné metódy.

Cieľom tejto práce bolo naštudovať možnosť interakcie 3D modelov vytvorených v programe Autodesk 3D Studio Max s technológiou Java Monkey Engine a demonštrovať toto prepojenie vytvorením trojrozmernej hry v prostredí Java SE. Knižnica jME obsahuje sedem herných typov, z ktorých je možné aplikáciu vyvíjať [13]. Každý z nich má svoje vlastné špecifikácie, výhody a nevýhody.

- `BaseGame` – poskytuje otvorenú, prístupnú a vysokorýchlostnú hernú slučku, kde každá jej iterácia je spracovaná tak rýchlo ako to môže CPU, prípadne GPU zvládnuť. Veľká časť z realizácie implementácie aplikácie je ponechaná na samotnom užívateľovi, čo mu umožňuje naplno využiť možnosti jME a svojej predstavivosti.
- `SimpleGame` – poskytuje už preddefinované koreňové uzly, ovládanie a nastavenia kamery, preto užívateľ nemá takú voľnosť pri vytváraní aplikácie.
- `StandardGame` – tento herný typ umožňuje vytvárať herné stavy, medzi ktorými môže prepínať, pracovať v OpenGL vlákne a obsahuje podporu pre tieňovanie objektov na scéne.
- `SimpleHeadlessGame` – podobný herný typ ako `SimpleGame`, ale navyše poskytuje lepšiu podporu `Swing/AWT`.
- `FixedFramerateGame` – tento herný typ dovoľuje nastaviť rýchlosť vykresľovania medzi jednotlivými snímkami.
- `FixedLogicrateGame` – hlavnej hernej slučke poskytuje iba určitý čas na aktualizáciu, zatiaľ čo vykresľovanie prebieha tak rýchlo ako to CPU, prípadne GPU dovoľuje.
- `VariableTimestepGame` – podobný herný typ ako `BaseGame`, ale navyše v sebe obsahuje časovač, ktorý meria čas medzi vykresľovaním jednotlivých snímkov.

7.1 BaseGame

Ja som sa rozhodol vytvoriť hru typu `BaseGame`, pretože ako je v úvode kapitoly popísané, poskytuje programátorovi väčšiu voľnosť pri implementácii hry.

`BaseGame` pozostáva z viacerých hlavných metód a to :

- `initSystem()`- toto je prvá metóda, ktorá sa zavolá po metóde `start()`, teda po spustení aplikácie. Slúži k inicializácii jME systému a k vytvoreniu zobrazovacieho okna, v ktorom sa bude všetko odohrávať.

- `initGame()`- metóda volaná po ukončení `initSystem()`. Spúšťa inicializáciu 3D grafu scény. Prebieha tu načítanie modelov, textúr, animácií a všetkých elementov používaných v aplikácii. Tie sú následne pripojené ku koreňovému uzlu.
- `update()`- zaisťuje aktualizáciu elementov v 3D scéne v čase. Metóda je volaná pri každej iterácii hernej slučky a je teda vykonávaná až do konca aplikácie.
- `render()`- zaisťuje vykresľovanie elementov do 3D scény v čase. Táto metóda je taktiež volaná pri každej iterácii.
- `reinit()`- metóda je volaná ak zobrazovací systém potrebuje znovu inicializovať, napríklad pri zmene rozlíšenia obrazovky.
- `cleanup()`- táto metóda je volaná po celkovom ukončení hernej slučky. A dôjde tu k vyčisteniu systému.
- `quit()`- je volaná tesne pred ukončením hlavnej slučky a spôsobí korektné vypnutie aplikácie.

8 Závodná hra v 3D

Táto kapitola sa venuje implementácii počítačovej hry. Popisuje vlastnosti elementov a ich dôležité prvky, z ktorých sa skladajú. Námety, na čo všetko je potrebné myslieť pri tvorbe hry a ako pri tom postupovať, som čerpal z [1].

Celá hra je vytvorená v hlavnej triede s názvom `Main`, ktorá dedí triedu `BaseGame` z jME knižnice. Po spustení aplikácie metódou `start()` dôjde k zobrazeniu úvodného okna s nastaveniami, kde je možné si zvoliť rozlíšenie okna, prípadne bitovú hĺbku farieb.

Následne je volaná metóda `initSystem()`, v ktorej sa zistia zadané nastavenia a vytvorí sa zobrazovacia plocha hry s týmito nastaveniami. Ďalej sa vytvorí kamera, ktorej sa nastaví uhol, výška a šírka záberu v závislosti od veľkosti displaya a vzdialenosť, po ktorú je kamera schopná dovidieť. Kamera sa nastaví ako renderovací prvok, teda užívateľ vidí iba to, čo vidí kamera. Potom sa vytvára časovač z triedy `LWJGLTimer()`, ktorý je potrebný pri aktualizáciách hernej slučky. Nakoniec dochádza ku napojeniu FengGUI [14] rozhrania do vytvorenej zobrazovacej plochy. FengGUI slúži pre vytvorenie grafického užívateľského rozhrania v hre.

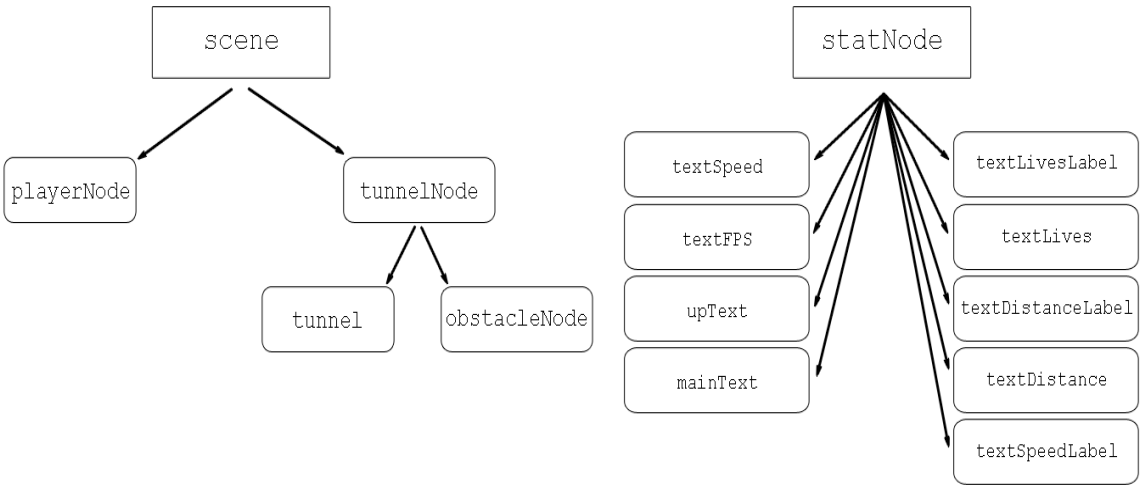
Po ukončení tejto metódy je volaná metóda `initGame()`. V nej sú vytvorené hlavné uzly hry. Pre dosiahnutie efektu zobrazenia priebehu načítania hry (viď. Obrázok 8.1: Priebeh načítania hry) je použitá trieda `Thread`, v ktorej dochádza k inicializácii objektov, modelov, textúr, zvukov a všetkých ostatných elementov vykresľovaných v scéne. Táto inicializácia je umiestnená vo vlákne, pretože vykresľovanie elementov sa začína až vo chvíli, keď dôjde k ukončeniu metódy `initGame()`. Tu však vznikal problém pri zobrazovaní priebehu, pretože pri automatickom volaní nasledujúcej metódy `update()` dochádzalo k chybám , lebo sa chceli aktualizovať aj objekty, ktoré ešte neboli vytvorené vo vlákne. Preto pre správne fungovanie bolo potrebné pozastaviť automatické iterácie metódy `update()` a to spôsobom vytvorenia dvoch herných stavov. Jedného pre úvodný priebeh načítania hry a druhý pre samotnú hru. K tomu slúži vytvorený `GameStateManager`, ktorý nastaví pri spustení aplikácie aktívny stav pre načítanie, čím umožní renderovanie a aktualizáciu iba pre seba. Teda na obrazovke je zobrazovaný priebeh načítania, zatiaľ čo v pozadí dochádza k načítavaniu. Po ukončení práce vo vlákne, teda po načítaní všetkých potrebných elementov, sa tento stav automaticky deaktivuje a tým povolí iterácie metód `update()` a `render()`.

Kde `update()` umožní aktualizácie vstupov z klávesnice, transformácií objektov, zvukového systém, systému pre detekciu kolízií a ostatných častí, ktoré je potrebné spúšťať pri každej iterácii. Metóda `render()` vykresľuje načítané elementy na scénu.

Obrázok 8.1: Priebeh načítania hry

8.1 Koreňové uzly

V hre sa vyskytujú dva hlavné uzly. A to `scene` a `statNode`, na ktoré sú pripájané všetky ostatné uzly. V hre sú vykresľované iba tieto dva uzly. Teda čo nie je k nim pripojené, to sa na scéne nevykreslí.



Obrázok 8.2: Graf scény koreňových uzlov

Ako je vidieť na obrázku ku koreňovému uzlu `statNode` sú pripojené iba jednoduché 2D texty, ktoré sa vytvárajú v metóde `buildText()`.

Tieto texty slúžia ako vizuálna reprezentácia informácii na hernej obrazovke (HUD - head-up display). Informujú užívateľa o základných informáciách, o ktorých by mal mať prehľad pri hraní hry. Ako napríklad dosiahnutá vzdialenosť, počet životov, aktuálna rýchlosť a o niektorých udalostiach, ku ktorým môže dôjsť v priebehu hry, napríklad ku kolízii.

Na uzol `scene` sú pripájane všetky ostatné uzly, ktoré už reprezentujú 3D objekty na scéne. Uzol `playerNode` je hlavným uzlom pre objekt vesmírnej lode a pre ďalšie uzly, ktoré s ním súvisia. Uzol `tunnelNode` je rodičovským uzlom pre uzly `tunnel`, ktorý je hlavným uzlom pre všetky časti objektu tunela, v ktorom sa vesmírna loď pohybuje a `obstacleNode`, ktorý je hlavným uzlom pre prekážky umiestnené v tuneli.

Uzlom `playerNode`, `tunnel` a `obstacleNode` sa táto práca bližšie venuje v ďalších častiach tejto kapitoly.

8.1.1 Spôsob vykresľovania

Pre uzol `scene` sú v hre vytvorené tri spôsoby vykresľovania. Prvým je `ZBufferState`, ktorý slúži k zobrazovaniu pixelov od pozície kamery až do vzdialenosti, po ktorú je kamera schopná vidieť. Teda objekty za touto hranicou nebudú vykresľované, pokiaľ neprídu bližšie k objektu kamery. Druhým je `CullState`, ktorý spôsobí, že zadné časti objektov nebudú zobrazované. A tretím je `FogState`, ktorý simuluje hmlu v scéne. Tá slúži pre estetickejšie vykresľovanie objektov, ktoré sa z diaľky približujú ku kamere. Objekty sa len tak naraz neobjavia na scéne, ale vyzerajú akoby vychádzali z hmly. Farba hmly je čierna, a preto nepôsobí rušivo v prostredí, pretože aj pozadie scény má čiernu farbu. Jej intenzita je nastavená na maximálnu hodnotu a to z dôvodu, aby nebolo vidieť hranicu, od ktorej sa objekty prestávajú vykresľovať.

Vďaka týmto spôsobom renderovania je zaručená väčšia rýchlosť hry a systém tak nie je zbytočne zaťažovaný, ako by tomu bolo bez tejto možnosti vylepšenia výkonu.

8.1.2 Osvetlenie

K vytvoreniu svetla dochádza v metóde `buildLightState()`, kde na celú scénu je aplikované priame svetlo `DirectionalLight`, ktoré svieti na všetky objekty z rovnakého uhlu. Preto je potrené nastaviť smer svetla metódou `setDirection`, jeho difúznu zložku pomocou metódy `setDiffuse`, ktorá určuje množstvo svetla, ktoré sú objekty schopné odraziť a v poslednom rade aj ambientnú zložku metódou `setAmbient`, ktorá udáva globálny farebný odtieň materiálov objektov.

Takto vytvorené svetlo je pripojené k stavu `LightState`, ktorý má na starosti obsluhu svetla a ten je následne priradený koreňovému uzlu `scene`. Týmto sa dosiahne už vyššie spomínaný efekt, že svetlo svieti rovnako na všetky objekty v scéne.

8.2 Zvukový systém

Zvukový systém je v prostredí jME veľmi ľahko implementovateľný. Pomocou volania jednej metódy `AudioSystem.getSystem()` sa jednoducho vytvorí. Pre dosiahnutie efektu priestorového 3D zvuku je potrebné systém pripojiť na pozíciu nejakého objektu v scéne, ktorý bude imitovať uši užívateľa. Najrozumnejším objektom v tomto prípade je objekt kamery.

Takto vytvorený systém podporuje načítanie dvoch formátov zvukového súboru a to wav alebo `ogg`. Ja som sa rozhodol pre formát `ogg`, pretože je nielen úspornejší z hľadiska miesta na disku, ale taktiež pri programovaní aplikácie dochádzalo niekedy k chybám pri prehrávaní formátov `wav`, čo malo za následok pád zvukového systému. Pri formáte `ogg` sa tieto chyby nevyskytovali. Všetky zvuky boli konvertované, upravované a prípadne vytvárané pomocou externého programu Fruity Loops Studio 9, ktorý ponúka veľkú škálu nastavení a efektov pre prácu so zvukom.

Pre načítanie zvuku slúžia dve metódy, ktoré rozlišujú o aký zvuk sa má jednať. Prvá `getSFX()` načítava zvuky pre špeciálne efekty, pridelí im pozíciu v scéne a po ich prehratí s výnimkou zvuku pre akceleráciu motora sa tieto zvuky neopakujú. Druhá `getMusic()` načíta zvuk, z ktorého sa vytvorí hudba do pozadia. Tento zvuk má nastavenú funkciu pre opakovanie.

Ako je spomenuté vyššie zvuk pre akceleráciu má nastavené opakovanie sa. To je z dôvodu docielenia zvuku motora, ktorý pracuje v určitých otáčkach a v závislosti od rýchlosti sa tieto otáčky zvyšujú, či znižujú. Preto je pre zvuk motora pridelených viac zvukov s rozdielnou výškou, čo má za následok prirodzený efekt zvuku zrýchľujúceho sa alebo spomaľujúceho sa motora.

Prehrávanie zvukov je zaobstarané jednoduchými metódami z triedy `AudioTrack`. Pre spustenie, zastavenie prípadne pozastavenie zvuku slúžia metódy `play()`, `stop()` a `pause()`.

V hre je implementovaný jeden zvuk, ktorý slúži ako hudba do pozadia hry. A viacero špeciálnych zvukových efektov, ktoré sú spustené pri nejakej udalosti. Napríklad po načítaní hry, počas akcelerácie vesmírnej lode, pri kolízii lode so stenami tunela a prekážkami, prípadne po zobratí bonusov, pri výbuchu prekážky, pri výbuchu lode, ak došlo k zníženiu počtu životov na nulu, pri stlačení menu tlačidla, prípadne pri prechode myšou nad ním alebo pri vystrelení lasera.

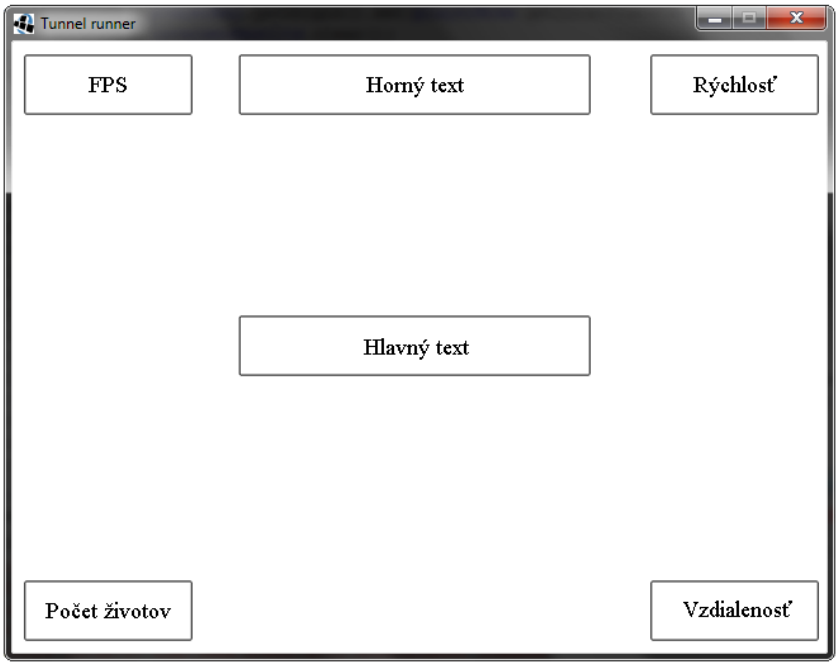
Aby sa predišlo chybám pri prehrávaní zvukov pre špeciálne efekty, je potrebné pred spustením zvuku overovať, či je daný zvuk spustený. Ak je, je nevyhnutné zvuk najprv zastaviť a následne opäť spustiť, ak nie je, stačí ho jednoducho spustiť. Taktiež je potrebné aktualizovať zvukový systém pri každej iterácii metódy `update()` a to pomocou metódy volanej na jej konci `AudioSystem.getSystem().update()`.

Týmto spôsobom predídeme chybovým výpisom, ktoré vznikajú pri opätovnom spustení už prehrávaného zvuku a následným pádom celého zvukového systému.

8.3 GUI

Grafické užívateľské rozhranie (GUI – Graphics User Interface) sa skladá z piatich hlavných častí.

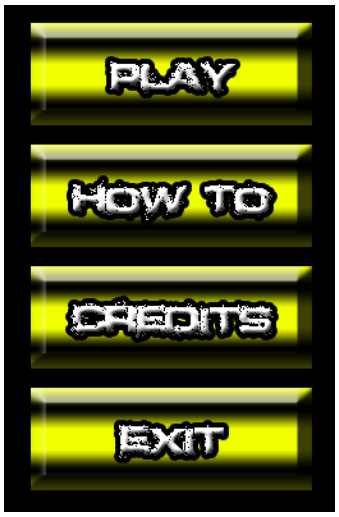
- Prvú reprezentujú 2D texty zobrazované na obrazovke počas hrania hry. Slúžia ako vizuálna reprezentácia informácii na hernej obrazovke (HUD), ktoré poskytujú užívateľovi základné informácie, o ktorých by mal mať prehľad počas hrania hry.



Obrázok 8.3: HUD

Text `textFPS` sa vykresľuje v ľavom hornom rohu a zobrazuje vykresľovaný počet snímkov za sekundu (FPS – frame per second). V pravom hornom rohu sú zobrazované `textSpeedLabel` a `textSpeed`, ktoré zobrazujú aktuálnu rýchlosť vesmírnej lode. V ľavom dolnom rohu `textLivesLabel` a `textLives` informujú o aktuálnom počte zostávajúcich životov. V pravom dolnom rohu `textDistanceLabel` a `textDistance` zobrazujú prejdenú vzdialenosť od štartu. Text `mainText` sa dočasne zobrazuje v strede obrazovky, napríklad pri narazení do stien tunela. Posledný `upText` sa nachádza v strede hornej časti obrazovky a informuje užívateľa počas celej dĺžky trvania nejakej udalosti. Napríklad pri získaní bonusu nezraniteľnosti sa vypíše text *invincible*.

- Druhú časť reprezentuje hlavné menu hry.



Obrázok 8.4: Hlavné menu hry

Ako vidieť na obrázku tlačidlo `Play` spúšťa hru. Tlačidlo `HowTo` podáva informácie o ovládaní hry. `Credits` zobrazí logo, informácie o hre a o jej autorovi. A posledné tlačidlo `Exit` hru vypne.

- Tretiu časť predstavuje menu počas pauzy hry (viď. Obrázok 8.5 – Pauzové menu hry). Je vyvolané prípadne zrušené stlačením klávesy `escape` na klávesnici. Pri jeho zobrazení dochádza k pozastaveniu iterácie hernej slučky `update()`, dochádza iba k aktualizáciám vstupov z klávesnice.

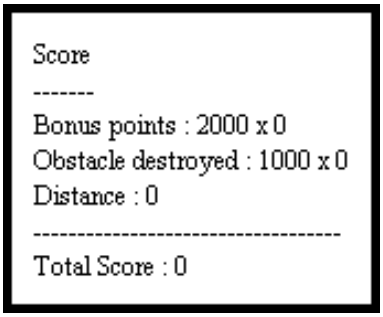
Obrázok 8.5: Pauzové menu hry

Tlačidlom `Resume` sa užívateľ dostane naspať do hry a opäť dochádza k iterácii `update()` metódy. Po stlačení tlačidla `Restart` sa hra reštartuje a znova sa inicializuje. Následne je volaná metóda `initGame()`. Pri stlačení `Quit to main menu` sa hra reštartuje a dostane sa do hlavného menu hry. Tlačidlo `Exit`, obdobne ako v hlavnom menu, aj tu spôsobí ukončenie aplikácie.

- Štvrtú časť predstavuje menu pre ovládanie zvukov. Skladá sa z tlačidla sfx a music. Po ich stlačení sa v hre vypne prípadne zapne prehrávanie hudby, či špeciálnych zvukových efektov.

Obrázok 8.6: Audio Menu

- Poslednú piatu časť tvorí tabuľka s nahraným počtom bodov. Výsledný počet bodov je súčet počtu zobraných bonusov, zostrelených prekážok, a prejdenej vzdialenosti.



Obrázok 8.7: Skóre tabuľka

8.3.1 FengGUI

Druhá až piata časť GUI hry je vytvorená pomocou externej knižnice FengGUI [14]. Jedná sa o grafické užívateľské rozhranie aplikačného programovacieho rozhrania založenom na OpenGL. FengGUI poskytuje všetky typické komponenty ako tlačidlá, rámce, zoznamy, textové oblasti a iné, ktoré sú potrebné pre vybudovanie kompletného GUI systému na vysokej úrovni, pri zachovaní vysokej efektivity pri jeho implementácii. Vzhľadom k tomu, že je založený na OpenGL, FengGUI sa vynikajúco hodí do multimediálneho, prípadne herného prostredia, pretože OpenGL umožňuje FengGUI byť rýchlym a zároveň príjemne estetickým grafickým rozhraním. Jeho ďalšie výhody pri spoločnej implementácie s javou sú:

- FengGUI je celý napísaný v Jave
- Je ľahko prispôsobiteľný vďaka štýlom v XML
- Je zameraný na výkon
- Podporuje hlavné OpenGL väzby pre Javu (JOGL, LWJGL)
- Jednoduché prepojenie s jME
- Umožňuje priame volanie OpenGL

Pre integráciu FengGUI s jME je potrebné najprv vytvoriť väzbu medzi jME zobrazovacou plochou a FengGUI metódou `org.fenggui.binding.render.lwjgl.LWJGLBinding()`. Po jej vytvorení je ďalej potrebné vytvoriť objekt `FengJMEInputHandler`, ktorý má na starosti obsluhovanie zobrazovacieho okna a zisťovanie či nedošlo ku stlačeniu kláves na klávesnici, prípadne na myši. Nasledujúcim krokom je už inicializácia komponentov. Keďže GUI v hre sa skladá z troch častí vytvorených prostredníctvom FengGUI, je potrebné vytvoriť dva objekty typu `Container()`, ktoré sú pripojené k už vytvorenému zobrazovaciemu oknu. Do prvého kontajnera sú následne pridané tlačidlá pre hlavné menu, prípadne tlačidlá pre pauzové menu. A do druhého tlačidlá pre spravovanie zvuku. Každé vytvorené tlačidlo má pre seba vytvorené dva objekty typu `eventListener` , kvôli zisťovaniu udalostí, či došlo ku stlačeniu alebo k vstupu myši ponad tlačidlo. A podľa typu udalosti dôjde k spusteniu zvukového efektu tlačidla spoločne s akciou, ktorá je pre tlačidlo naimplementovaná.

Pre vykresľovanie takto vytvorených komponentov v zobrazovacom okne je potrebne volať metódu `display()` v metóde `render()`, aby došlo k správnemu vykresleniu týchto GUI komponentov.

8.4 Uzol tunnel

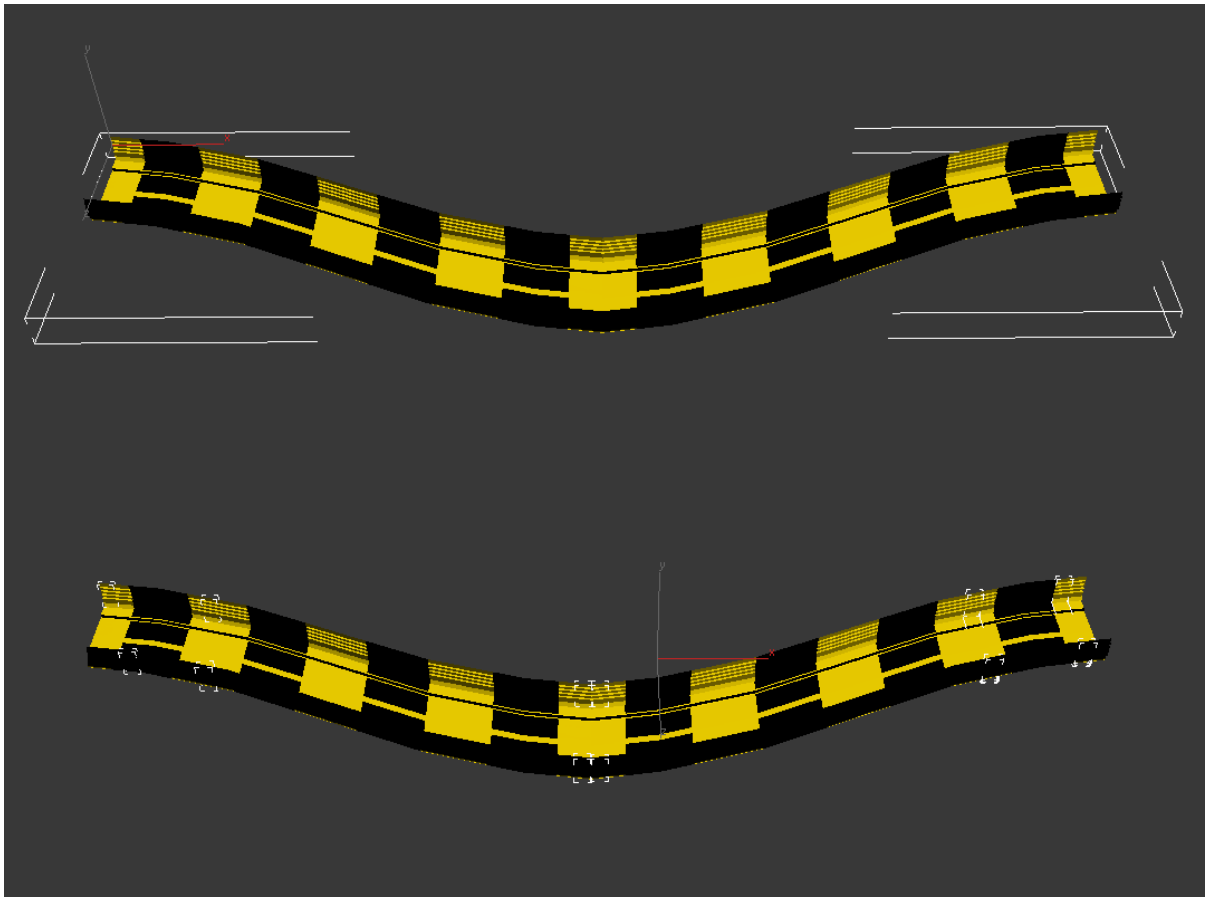
Jedným z hlavných objektov v scéne je objekt tunela. Uzol `tunnel` je rodičovským uzlom pre všetky časti tunelov. Tunel v hre predstavuje statický objekt a je tvorený z menších častí, ktoré na seba nadväzujú. Modely tunelov sú vytvorené prostredníctvom externého programu 3D Studio Max a sú exportované do formátu `3DS`.

8.4.1 Typy tunelov

V hre sa stretneme s piatimi typmi tunelov a to s rovným tunelom, tunelom zabáčajúcim doľava, tunelom zabáčajúcim doprava, tunelom stúpajúcim hore a tunelom klesajúcim dole. Všetky tieto časti sú ľahko nahraditeľné a prípadné rozšírenie spektra typov tunelov je pomerne jednoduché. Stačí dodržať dĺžku objektu tunela, veľkosť jeho začiatočného a koncového prierezu a tunel s takýmito parametrami sa dá jednoducho spojiť s predošlými typmi. Formát `3DS` je ľahko importovateľný do jME a to prostredníctvom triedy `ObjectLoader.java`, ktorá dokáže importovať model spoločne s textúrou. Podklady pre vytvorenie tejto triedy boli čerpané z [18].

Každý z modelov tunela nesie na sebe textúru čierno–žltej farby. Je to v podstate obrázok vytvorený v externom programe Adobe Photoshop CS4 vo formáte `JPG`. V programe 3D Studio Max bol model tunela nielen vytvorený, ale bola mu aj pridelená táto textúra. A vďaka formátu `3DS`, ktorý v sebe dokáže niesť vlastnosti o pozícii, type textúry, veľkosti modelu, bola práca s importovaním modelu do scény pomerne jednoduchá.

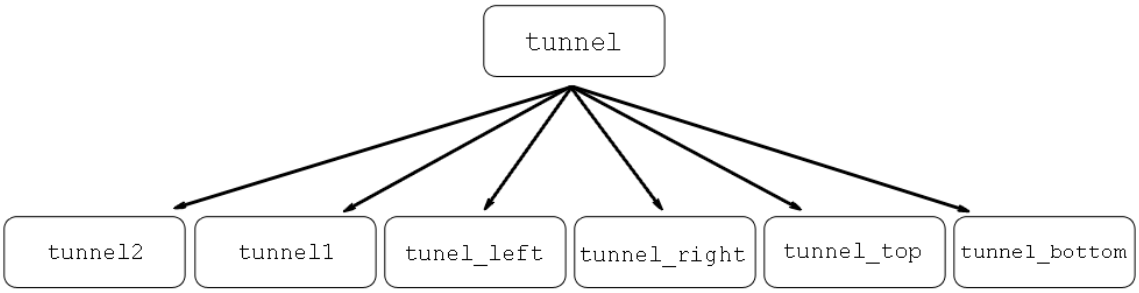
Avšak z dôvodu zisťovania kolízií bolo nutné každý model tunelu rozdeliť na jednotlivé steny, teda hornú, dolnú, ľavú a pravú. A tieto steny ešte rozdeliť na menšie časti (viď. spodný obrázok u Obrázok 8.8: Hranice tunela). Pretože ak by sme chceli importovať celý tunel naraz , jME by takto načítanému modelu pridelila hranice (Bounding Volume) objektu, akoby sa jednalo o vyplnený obdĺžnik (viď. horný obrázok u Obrázok 8.8: Hranice tunela). Toto je však v našom prípade nežiaduci efekt, pretože pri zisťovaní kolízií, by k nim dochádzalo aj vtedy, kedy by sa vesmírna loď nedotýkala stien tunela, ale nachádzala by sa vo vnútri týchto vymedzených hraníc. Preto bolo nutné tieto modely do scény importovať po týchto menších častiach a následne z nich poskladať každý typ tunela do výsledného tvaru. Vďaka vyššie spomenutej vlastnosti `3DS` formátu a to, že si uchováva v sebe informáciu o polohe modelu, toto skladanie tunela nebolo až tak časovo náročné.



Obrázok 8.8: Hranice tunela

8.4.2 Hierarchia grafu

Hierarchiu grafu hlavného uzlu `tunnel` je vidieť na obrázku.



Obrázok 8.9: Graf scény tunela

Kde listy stromu reprezentujú typy tunelov , ktoré je možné spájať.

8.4.3 Generovanie tunela

Ako je spomenuté vyššie celkový výsledný tunel je poskladaný z viacerých typov, ktoré na seba svojou štruktúrou dokážu nadväzovať. K tomuto účelu slúži metóda `generateTunnel()`, ktorá je volaná vždy, keď vesmírna loď prekročí polovicu dĺžky daného typu tunela, v ktorom sa práve nachádza. Vďaka čomu sa na koniec tejto časti pripojí ďalšia časť tunela. Časti na seba nenadväzujú vždy v rovnakom poradí, ale sú v tejto metóde generované náhodne.

Pomocou metódy `FastMath.nextRandomFloat()`, ktorá zo zvoleného intervalu dokáže náhodne vybrať číslo, je generovaný typ tunela, ktorý bude nasledovať, vybraný náhodne. Pre napájanie tunelov za seba je preto ešte potrebné vedieť jeho dĺžku, aby sa korektne napojil za predchádzajúcu časť tunela.

Vďaka takémuto spôsobu generovania tunela, je výsledný tunel vždy originálnym dielom, čo určite prispeje k hrateľnosti a väčšiemu zážitku z hry.

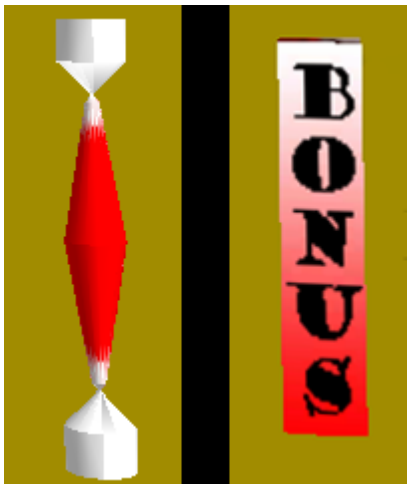
8.5 Uzol obstacleNode

Ďalším z hlavných uzlov je `obstacleNode`, ktorý je rodičovským uzlom pre všetky typy prekážok v tuneli. Tie v hre predstavujú taktiež statické objekty. Modely prekážok sú vytvorené prostredníctvom externého programu 3D Studio Max a sú exportované do formátu `3DS`.

8.5.1 Typy prekážok

V hre sa stretneme s dvoma typmi prekážok. Prvým je prekážka, do ktorej ak narazí vesmírna loď, tak dôjde k zníženiu počtu životov. Druhým typom je prekážka vo forme bonusu, ktorá v sebe ukrýva vylepšenie nejakej vlastnosti objektu lode. Taktiež aj tieto modely, je možné nahradiť iným modelom, čo otvára možnosť pre užívateľa si vytvoriť vlastné modely prekážok. Samozrejme je potrebné v rozumných medziach zachovať ich rozmery.

Na prekážky je nanesená textúra bielo-červenej farby, aby boli dobre viditeľné. Pre prekážku typu bonus tu je textúra doplnená o nápis *bonus*.



Obrázok 8.10: Typy prekážok

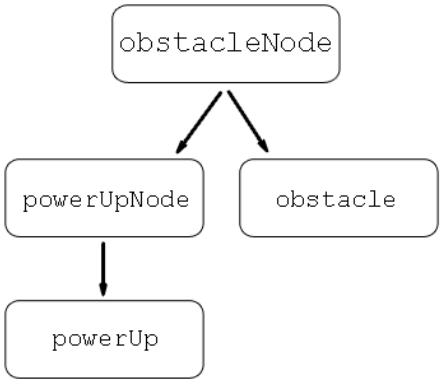
8.5.2 Typy bonusov

V hre sú implementované dva typy bonusov. Prvým je bonus, ktorý zvyšuje počet životov o jeden, avšak nikdy neprevýši maximálny počet životov, ktorý mala loď na začiatku hry. Druhým je bonus nezraniteľnosti na určitú dobu, takže užívateľ sa nemusí obávať straty života pri kolízii s prekážkami počas trvania tohto bonusu.

Metóda pre zobratie bonusu `pickPowerup()`, je navrhnutá tak aby užívateľ nevedel, ktorý bonus sa chystá zobrať a z tohto dôvodu majú všetky prekážky typu bonus rovnaký tvar aj textúru. Pri kolízii lode s bonusom dostane teda hráč náhodný typ bonusu. Táto metóda je ľahko rozšíriteľná, čím vzniká možnosť pomerne jednoducho doimplementovať ďalšie typy bonusov do hry.

8.5.3 Hierarchia grafu

Hierarchiu grafu hlavného uzlu `obstacleNode` je vidieť na obrázku.



Obrázok 8.11: graf scény prekážok

Kde `obstacleNode` združuje všetky typy prekážok. Na uzol `powerUpNode` sú pripájané iba bonusové typy prekážok.

8.5.4 Generovanie prekážok

Pri návrhu prekážok pre hru bol kladený dôraz na zlepšenie hrateľnosti hry, a preto sú objekty prekážok do tunela generované v náhodnom počte metódou `generateObstacle()`, ktorá sa volá toľkokrát, koľko sa má prekážok vygenerovať do danej časti tunela. Toto číslo sa taktiež generuje náhodne, takže v každej časti tunela je iný počet prekážok. Taktiež sú tieto prekážky náhodne rozmiestnené v tuneli pomocou metódy `randomize()` z triedy `Obstacle.java`. A to nasledujúcim spôsobom. Najprv sa náhodne vyberie x-ová súradnica polohy prekážky v tuneli a potom jeho náhodná z-ová súradnica. Pre takto vytvorenú prekážku sa potom zisťuje, či je v správnej výške, teda či jeho y-ová súradnica odpovedá výške tunela v danom bode, kde je prekážka umiestnená. Správna y-ová súradnica sa zistí pomocou metódy `calculatePick()`, ktorá na základe vstupných parametrov (x-ovej a z-ovej súradnice) polohy danej prekážky, dokáže nájsť priesečník medzi myslenou čiarou smerujúcou kolmo nadol z prekážky a spodného dielu tunela, ktorý predstavuje podlahu. Tento priesečník sa dá zistiť metódou `getTriangle()`, ktorá vráti požadovaný vertexový trojuholník podlahy tunela obsahujúci tento priesečník. Následne už len stačí priradiť hodnotu y-ovej súradnice trojuholníku pre danú prekážku a tým sa docieli výsledná poloha prekážky, ktorá bude položená na podlahe tunela. Ešte je potrebné overiť či prekážka nemá kolíziu s bočnými stenami tunela, prípadne či sa nenachádza mimo podlahy tunela. Ak teda všetko prebehlo úspešne prekážka sa bude nachádzať vo vnútri, bez dotyku s bočnými stenami a na podlahe tunela.

Tento algoritmus sa opakuje pre každú novú generovanú prekážku a navyše ešte dochádza k posunutiu náhodného intervalu x-ovej a z-ovej súradnice o veľkosť predchádzajúcej prekážky, aby nedošlo ku kolízii, či prekrývaniu prekážok navzájom.

Všetky prekážky sú vytvárané zo samostatnej triedy `Obstacle.java`, ktorá dedí triedu `SharedNode`, a tým sa docieli zdieľanie originálneho modelu 3D objektu pre všetky novo vygenerované prekážky a taktiež aj úspora pamäte, pretože nie je potrebné 3D model pre každú prekážku znovu načítavať, ale stačí to urobiť iba raz pri inicializácii hry.

8.5.5 Generovanie bonusov

Pre generovanie bonusov v hre je použitý rovnaký princíp ako u prekážok, avšak s tým rozdielom, že bonus je generovaný iba raz pri každom vytvorení zhluku prekážok v danej časti tunela. V podstate dôjde k nahradeniu jednej náhodnej prekážky bonusom, ktorý zaberie jej pozíciu v scéne.

8.5.6 Osvetlenie

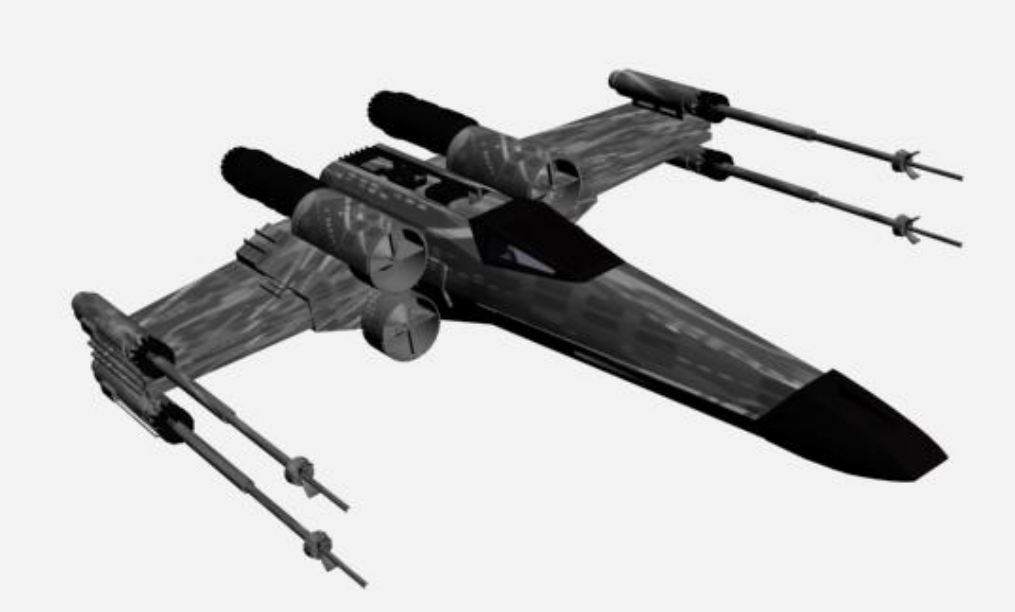
K vytvoreniu osvetlenia prekážky dochádza v triede `Obstacle.java`, kde je pre danú prekážku vytvorené svetlo `PointLight`, ktoré svieti priamo na prekážku. Nastavením vlastností svetla sa docieli krajšie a výraznejšie zobrazenie objektu prekážky v scéne.

Takto vytvorené svetlo je pripojené k stavu `LightState`, ktorý má na starosti obsluhu svetla a ten je následne priradený ku každej vytváranej prekážke. Týmto sa dosiahne efekt, že každá prekážka má svoje vlastné svetlo.

8.6 Uzol playerNode

Posledným z hlavných objektov v scéne je objekt vesmírnej lode. Uzol `playerNode` je rodičovským uzlom pre všetky časti lode, z ktorých je tento model zložený. Loď v hre predstavuje jediný dynamický objekt. Model lode je prevzatý z [15] a následne upravený prostredníctvom externého programu 3D Studio Max a exportovaný do formátu `3DS` (viď. Obrázok 8.12: Model vesmírnej lode).

Model vesmírnej lode nesie na sebe textúru so sivým vojnovým motívom, ktorá je roztiahnutá po celom povrchu lode. Niektoré časti ako napríklad špic lode alebo zadné turbíny majú čiernu textúru kvôli lepšiemu estetickému vzhľadu. Aj tieto textúry ako aj predošlé u iných modelov sú v podstate obrázky vytvorené v externom programe Adobe Photoshop CS4 vo formáte `JPG`.



Obrázok 8.12: Model vesmírnej lode

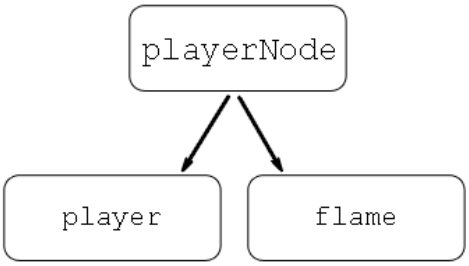
Objekt vesmírnej lode sa spolu s jeho metódami nachádza v triede `PlayerNode.java`, v ktorej sa spravuje chovanie 3D modelu vesmírnej lode v prostredí tunela. Preto bolo potrebné implementovať metódy pre nastavenie výzoru modelu, ovládanie rýchlosti a akcelerácie vpred prípadne do strán, brzdenia, maximálnej a minimálnej povolenej rýchlosti vpred a do strán a hlavnú metódu v tejto triede `update()`, ktorá má na starosti aktualizáciu pohybu lietadla, po každom vykreslení snímku na scéne. Základne vlastnosti objektu vesmírnej lode sú nastavované v konštruktore triedy, ktoré pomerne jednoducho umožňujú meniť jeho jazdné vlastnosti.

8.6.1 Ovládanie

Pre ovládanie vesmírnej lode je najprv potrebné vytvoriť objekt z triedy `PlayerHandler`, ktorý dedí triedu `InputHandler` a tento objekt je aktualizovaný pri každej iterácii hernej slučky. Týmto sa docieli možnosť stláčania viacerých kláves naraz, pričom hra je schopná na ne súčasne reagovať. Tým je napríklad umožnené súčasné pridávanie rýchlosti a natáčanie lode do strán. V triede `PlayerHandler` sú vytvorené objekty, na ktoré sú namapované klávesy príslušných akcií. Pri stlačení klávesy sa vyvolá jej odpovedajúca akcia a vytvorí sa objekt triedy `MovementAction`, ktorá dedí triedu `KeyInputAction` pre túto akciu. Reagovanie na ňu obsluhuje metóda `performAction()`, v ktorej sa vyvolá určitý úkon v závislosti od stlačenej klávesy na klávesnici.
Takýmto spôsobom je v hre implementovaný pohyb vesmírnej lode smerom dopredu, jej brzdenie, zabáčanie a zároveň nakláňanie do strán. K tomu slúžia namapované klávesy šípka dopredu pre akceleráciu, šípka dozadu pre brzdenie, šípka doľava pre zabočenie a naklonenie smerom doľava a nakoniec šípka doprava pre zabočenie a naklonenie smerom doprava. Pri uvoľnených šípkach vľavo a vpravo dochádza k vyrovnaniu naklonenia vesmírnej lode.

8.6.2 Hierarchia grafu

Hierarchiu grafu hlavného uzlu `playerNode` je vidieť na obrázku.



Obrázok 8.13: Graf scény vesmírnej lode

Kde uzol `player` predstavuje 3D objekt vesmírnej lode ako taký a uzol `flame` slúži pre dosiahnutie efektu horiaceho motora.

8.6.3 Transformácie

Na model vesmírnej lode sú aplikované nasledujúce transformácie :

- Pohyb vesmírnej lode
- Rotácia vesmírnej lode
- Zmena veľkosti modelu vesmírnej lode

8.6.3.1 Pohyb vesmírnej lode

Pohyb vesmírnej lode je aplikovaný na jej hlavný rodičovský uzol `playerNode`, ktorý sa môže pohybovať v troch smeroch a to po lokálnych osiach X, Y a Z.

V smere po z-ovej súradnici, čiže smerom dopredu sa vesmírna loď pohybuje na základe jej rýchlosti vynásobenej časom medzi vykreslením snímkov kvôli optimalizácii pri zmene rýchlosti vykresľovania snímkov.

```
public void accelerate(float time) {
      velocity += time * acceleration;
      if (velocity > maxSpeed) {
            velocity = maxSpeed;
      }
}
```

Kde `time` predstavuje čas medzi vykreslením snímkov, `acceleration` rýchlosť akcelerácie vesmírnej lode a `velocity` jej celkovú rýchlosť, o ktorú sa bude loď pohybovať smerom po z-ovej

súradnici. Jeho maximálna rýchlosť je obmedzená na hodnotu prislúchajúcej `maxSpeed`. Tento pohyb dopredu sa vykoná vďaka metóde `this.localTranslation.` `.addLocal(this.localRotation.getRotationColumn(2, tempVa).` `.multLocal(velocity * time))` z triedy `PlayerNode.java`, ktorá k predošlej pozícii na z-ovej osi pripočíta násobok rýchlosti a času. Táto metóda sa vykonáva pri každej iterácii hernej slučky.

Pre spomalenie rýchlosti prípadne úplné zabrzdenie slúži metóda :

```
public void brake(float time) {
      velocity -= time * braking;
      if (velocity < minSpeed) {
            velocity = minSpeed;
      }
}
```

Táto metóda funguje na rovnakom princípe ako metóda pre pridávanie rýchlosti, akurát s tým rozdielom, že od celkovej rýchlosti odpočítava hodnotu prislúchajúcu brzdeniu `braking`. A nikdy nespomalí viac ako je hodnota prislúchajúca pre `minSpeed`.

Pre pohyb po x-ovej súradnici, čiže smerom doľava prípadne doprava sa vesmírna loď pohybuje na základe jeho bočiacej rýchlosti vynásobenej časom medzi vykreslením snímkov. Taktiež kvôli optimalizácii pri zmene rýchlosti vykresľovania snímkov. K pohybu do strán slúžia tri metódy a to:

```
public void strafeLeft(float time) {
      if (ignore){
            strafemove += time * strafeSpeed*20 ;
            if (strafemove <prevstrafemove || strafemove > 0){
                  if (strafemove > maxStrafeSpeed){
                  strafemove = maxStrafeSpeed;
            }
      }else strafemove = 0;
      if (strafemove!= 0)prevstrafemove = strafemove;
      }
}
```

Táto metóda umožňuje lodi zabáčať doľava. Kde `time` predstavuje čas medzi vykreslením snímkov, `strafeSpeed` akceleráciu bočiacej rýchlosť vesmírnej lode a `strafemove` jeho celkovú bočiacu rýchlosť, o ktorú sa bude loď pohybovať smerom po x-ovej súradnici. Jeho maximálna bočiaca rýchlosť je obmedzená na hodnotu `maxStrafeSpeed`.

Tento pohyb doľava sa vykoná vďaka metóde `this.localTranslation.` `.addLocal(this.localRotation.getRotationColumn(0, tempVa).` `.multLocal(strafemove * time))` z triedy `PlayerNode.java`, ktorá k predošlej pozícii na x-ovej osi pripočíta násobok bočiacej rýchlosti a času. Táto metóda sa taktiež vykonáva pri každej iterácii hernej slučky.

Pre zabáčanie doprava slúži metóda :

```
public void strafeRight(float time) {
      if(ignore){
            strafemove -= time * strafeSpeed*20 ;
            if (strafemove >prevstrafemove || strafemove < 0){
                  if (strafemove < -maxStrafeSpeed){
                        strafemove = -maxStrafeSpeed;
            }
      }else strafemove =0;
      if (strafemove!= 0)prevstrafemove = strafemove;
      }
}
```

Ktorá funguje na rovnakom princípe ako metóda pre zabočenie doľava, akurát sa líšia v znamienku pre bočiacu rýchlosť `strafemove`, aby bolo možné zabáčať doprava.

Pre pohyb po y-ovej súradnici, čiže smerom hore, prípadne dole sa vesmírna loď pohybuje na inom princípe ako tomu bolo pri pohybe dopredu a do strán. Pohyb po y-vej súradnici sa vypočítava podobným spôsobom ako tomu bolo pri zisťovaní správnej y-ovej súradnice pre umiestnenie prekážky do tunela v správnej výške. Čiže je potrebné zistiť priesečník medzi spodnou častou tunela a myslenou kolmicou vedenou smerom nadol z objektu vesmírnej lode. Vyššie spomínanou metódou `calculatePick()` a `getTriangle()`, dostaneme požadovaný vertexový trojuholník, u ktorého zistíme y-ovu súradnicu, ktorú následne priradíme pre y-ovú súradnicu vesmírnej lode navýšenú o číslo 150. Týmto sa dosiahne efekt vznášania sa vesmírnej lode nad podlahou tunela a umožní jej stúpať prípadne klesať, v závislosti od typu tunela, v ktorom sa práve nachádza.

8.6.3.2 Rotácia vesmírnej lode

Rotácia vesmírnej lode je aplikovaná na jej hlavný rodičovský uzol `playerNode`, ktorý môže rotovať v troch smeroch a to po lokálnych osiach X, Y a Z. Rotácia je aplikovaná na tento uzol, pretože je potrebné, aby tieto rotácie dedili aj jeho potomkovia.

O rotáciu v smere po y-ovej, čiže otáčanie a z-ovej súradnici, čiže nakláňanie do strán vesmírnej lode je vykonávane v metóde `updatePlayer()`, ktorá je volaná pri každej iterácii hernej slučky. K rotácii po týchto osiach dochádza iba pri zabáčaní vesmírnej lode.

```
angleZ += (time * 1.0f);
angleY += (time * 0.75f);
if (angleZ > 0.75) {
     angleZ = 0.75f;
}
if (angleY > 0.5) {
     angleY = 0.5f;
}
rotQuat1.fromAngles(0, -angleY, angleZ);
playerNode.setLocalRotation(rotQuat1);
```

Kde `angleY` predstavuje rotačný uhol pre y-ovú os a `angleZ` rotačný uhol pre z-ovú os. Čiže ak má vesmírna loď kladnú bočiacu rýchlosť, teda zabáča doľava, k rotačným uhlom sú počas doby zabáčania pričítavané kladné hodnoty vynásobené časom medzi vykreslením snímkov kvôli optimalizácii pri ich zmene rýchlosti vykresľovania. Pre tieto rotačné uzly sú nastavené maximálne hodnoty, aby nedochádzalo k nežiaducim účinkom ako napríklad úplné otočenie, či ku preklopeniu lode. Rotačný kvaternion [16] `rotQuat1`, je potom vypočítaný z týchto rotačných uhlov `angleY` a `angleZ` a ten je následne aplikovaný na rodičovský uzol `playerNode`.

Rotácia na druhú stranu, teda pri zápornej bočiacej rýchlosti a zabáčaní doprava je vykonávaná obdobným spôsobom, akurát s jediným rozdielom, že rotačným uzlom `angleY` a `angleZ` sú odrátavané hodnoty a všetky hodnoty sa zmenia z kladných na záporné. Pokiaľ loď nezabáča, má teda nulovú bočiacu rýchlosť, je potrebné, aby sa naklonenie do strán normalizovalo do pôvodnej polohy.

```
if (angleZ<0){
     angleZ += (time * 1.0f);
     if (angleZ >0) angleZ=0;
}
if (angleZ>0){
     angleZ += (-time * 1.0f);
     if (angleZ <0) angleZ=0;
}
rotQuat3.fromAngles(0, -angleY, angleZ);
playerNode.setLocalRotation(rotQuat3);
```

Kde sa rotačný uzol `angleZ` postupne privedie k nulovej hodnote. Postupne preto, aby nedošlo k neprirodzenému vyrovnaniu naklonenia lode. Avšak uhlu `angleY` je ponechaná predchádzajúca hodnota, pretože ponechanie otočenia lode v y-ovej osi, je požadovaný efekt, ktorý jej umožňuje sa pohybovať dopredu aj v smere natočenia predného trupu lode.

O rotáciu v smere po x-ovej, čiže nakláňanie vesmírnej lode dopredu, prípadne dozadu je taktiež vykonávané v metóde `updatePlayer()`, s tým rozdielom, že tam vystupuje iba jeden rotačný uhol `angleX` a k rotácii po tejto osi dochádza iba pri prelete lode cez klesajúci alebo stúpajúci model tunela. Z tohto dôvodu je potrebné si uchovávať predošlú polohu vesmírnej lode na y-ovej osi, aby bolo možné zistiť, pri porovnaní s aktuálnou polohou na y-vej osi, či loď klesá, prípadne stúpa. A v závislosti od toho nakloniť trup lode dopredu respektíve dozadu.

```
angleX += (-time * 0.75f );
if (angleX < -0.12) {
     angleX = -0.12f;
}
rotQuat5.fromAngles(angleX, -angleY, angleZ);
playerNode.setLocalRotation(rotQuat5);
```

Rotačný kvaternion je vypočítavaný zo všetkých rotačných uhlov, aby bolo možné trup lode nakláňať vo všetkých osiach zároveň. Pretože loď môže bočiť a nakláňať sa do strán aj pri stúpaní respektíve klesaní.

8.6.3.3 Zmena veľkosti modelu vesmírnej lode

Keďže exportovaný model vesmírnej lode je príliš veľký oproti ostatným modelom na scéne, bolo ho potrebné zmenšiť na jednu štvrtinu jeho pôvodnej veľkosti.

8.6.4 Kamera

Veľmi dôležitým faktorom, ktorý ovplyvňuje dojem z hry je zobrazovanie scény. S tým súvisí nastavenie vlastností kamery, a preto je dôležité nájsť vhodný pohľad na scénu. Knižnica jME v sebe obsahuje triedu `ChaseCamera`, ktorá umožňuje sledovanie zadaného cieľa, v tomto prípade objektu vesmírnej lode, s určitými nastaveniami, ktoré ovplyvňujú spôsob jeho sledovania.

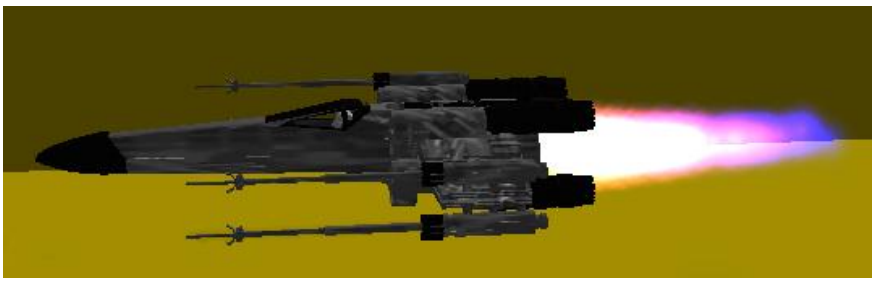
V hre je objekt kamery vytvorený v metóde `buildChaseCamera()`, kde sú tejto kamere nastavené parametre prenasledovania objektu vesmírnej lode. Tieto parametre sú zadané v hashovacej mape a tá je predaná objektu kamery. Prvým a základným nastavením je nastavenie výšky kamery nad sledovaným objektom. Pre dosiahnutie pohľadu spoza lode, je kamera umiestnená dvaapolkrát vyššie ako je výška lode. Takže kamera sleduje objekt mierne s nadhľadom. Ďalším nastavením je nastavenie vzdialenosti, z ktorej môže objekt kamery sledovať zadaný cieľ. Táto vzdialenosť v závislosti od nastavení sa môže zväčšovať prípadne zmenšovať. Ovládanie približovania a odďaľovania je nastavené na koliesko myši. Ďalším nastavením je možnosť, aby kamera reagovala na rýchlosť vesmírnej lode. Teda čím rýchlejšie sa pohybuje cieľ, tým sa vzdialenosť medzi kamerou a cieľom zväčšuje a naopak. Navyše pohybmi počítačovej myši je možné kamerou rotovať okolo sledovaného cieľa. To ako bude kamera rotovať je možné taktiež nastaviť. Pre korektné zobrazovanie

je potrebné aktualizovať objekt kamery s každou iteráciou hernej slučky a k tomu slúži metóda `chasecam.update()`.

Všetkými týmito nastaveniami parametrov pre kameru je možné jednoducho zlepšiť dojem užívateľa z pohľadu na objekty v scéne a tým zaručene prispieť k atraktívnosti hry.

8.6.5 Efekt motora

Pre dosiahnutie efektu horiaceho motora (viď. Obrázok 8.14: Efekt motora) za predpokladu, že sa vesmírna loď pohybuje, je k rodičovskému uzlu `playerNode` pripojený uzol `flame`.



Obrázok 8.14: Efekt motora

K vytvoreniu tohto efektu je potrebné implementovať `ParticleFactory`, čo je systém pre tvorbu čiastočiek, ktoré budú v podstate simulovať vychádzajúci plameň zo zadnej turbíny vesmírnej lode. Prvým krokom je vytvorenie objektu z triedy `ParticleFactory` metódou `buildParticles()`, ktorá nám vytvorí požadovaný počet čiastočiek. Následne je im potrebné nastaviť chovanie. Smer vypúšťania čiastočiek, ich počiatočnú a koncovú veľkosť, z akej pozície sa majú vypúšťať, ich minimálnu a maximálnu životnosť v scéne, počiatočné a koncové farebné spektrum.

Potom je im ešte nastavený spôsob vykresľovania v scéne a to `BlendState`, ktorý týmto čiastočkám vytvorí transparentnosť a textúru pre tieto čiastočky, čo docieli ich vierohodnejší výzor a krajší vzhľad. Takto vytvorené čiastočky sú pripojené k uzlu `flame`.

8.6.6 Strieľanie

Pre väčší zážitok z hry je v aplikácii implementovaná možnosť streľby. K tomu slúži klávesa medzerník na klávesnici. Po jej stlačení dôjde k vytvoreniu objektu náboja v internej triede `FireBullet`. Tomuto náboju je pridelený vlastný materiál červenej farby, aby bol dobre viditeľný v scéne a nastavené hranice náboja, aby bolo možné detekovať kolízie objektov s týmto nábojom. Na to slúži v jME metóda `setModelBound()`. Po vytvorení objektu náboja je pre náboj následne vytvorený kontrolér internou metódou `BulletUpdater`, ktorá dedí triedu `Controller` a riadi pohyb náboja v scéne od jeho vytvorenia až po jeho zánik. V nej je preto potrebné implementovať `update()` metódu, ktorá sa vyvoláva pri každej iterácii hernej slučky počas životnosti strely. Preto

je v tejto triede náboju pridelená dĺžka životnosti, po ktorej strela zanikne, smer vystreleného náboja a jeho rýchlosť. Rýchlosť je nastavená na dvojnásobok maximálnej rýchlosti modelu vesmírnej lode. Smer je dopredu od lode po z-ovej osi a dĺžka životnosti je nastavená na 5 sekúnd.

8.6.7 Detekcia kolízií

Jedným z dôležitých faktorov, na ktoré treba myslieť pri vytváraní 3D hry je interakcia objektov v scéne. Preto je v hre implementovaná metóda `processCollision()`, ktorá slúži na detekciu kolízii medzi objektmi. Prvou možnou situáciou je kolízia medzi tunelom a prekážkami, ktorá slúži pre správne umiestnenie prekážok do tunela. Týmto problémom sa zaoberala podkapitola 8.5.4 Generovanie prekážok. V hre sa však stretneme s ďalšími situáciami vzájomnej interakcie.

8.6.7.1 Kolízia vesmírnej lode s tunelom

Pre udržanie vesmírnej lode vo vnútri tunela je potrebné kontrolovať ich vzájomné kolízie. K tomu slúži volanie metódy z knižnice jME `player.hasCollision(tunnel,false)`, kde `player` predstavuje objekt vesmírnej lode a `tunnel` objekt celého tunela. Toto overovanie kolízie prebieha s každou iteráciou hernej slučky. A ak teda dôjde ku kolízii medzi týmito objektmi je následne volaná metóda `processCollision()`, v ktorej sa zisťuje, s ktorou stenou tunela sa loď dotýka. Ak sa jedná o ľavú stenu tunela, je pri náraze na túto stenu vesmírna loď odrazená smerom doprava a jej aktuálna rýchlosť sa zníži. Obdobne je tomu pri náraze do pravej steny tunela, kde je loď odrazená doľava. Ako je spomenuté v podkapitole 8.6.3.1 Pohyb vesmírnej lode, ku kolízii lode s hornou alebo dolnou časťou tunela dôjsť nemôže z dôvodu, že vesmírna loď sa vždy nachádza nad spodnou častou tunela v určitej výške. Pri kolíziách je užívateľ upozornený nápisom, že došlo k nárazu a je prehraný zvukový efekt pre kolíziu.

8.6.7.2 Kolízia vesmírnej lode s prekážkami

Pre tento typ kolízie slúži volanie metódy `player.hasCollision(obstacleNode,false)`, kde `player` predstavuje objekt vesmírnej lode a `obstacleNode` uzol združujúci všetky typy prekážok. Následne je volaná metóda `processCollision()`, v ktorej sa zisťuje, s akým typom prekážky došlo ku kolízii. Ak sa jedná o normálnu prekážku, pri náraze na ňu je odrátaný život vesmírnej lodi, ktorá sa potom na určitú dobu stane nezraniteľnou, číže po tento čas nemôže dôjsť k ďalšiemu nárazu a daná prekážka je odstránená zo scény, čo sprevádza efekt vybuchnutia spolu so zvukovým efektom. Tento efekt je podobne tvorený ako plameň z motora, čiže systémom čiastočiek s textúrou. Ak sa jedná o bonusovú prekážku, pri náraze na ňu je prirátaný typ bonusu pre vesmírnu loď. Taktiež sa táto prekážka odstráni zo scény, ale iba za sprievodu zvukového efektu, ktorý prislúcha danému typu bonusu.

8.6.7.3 Kolízia náboja s prekážkami

Pri tomto type kolízie sa postupuje obdobne ako pri náraze vesmírnej lode na prekážku, teda ak sa jedná o normálnu prekážku, tak dôjde k jej vybuchnutiu a náboj sa odstráni zo scény. Akurát nedôjde k odrátaniu života vesmírnej lode. Ak náboj narazí na bonusovú prekážku, je taktiež odstránený zo scény a vesmírna loď získa daný bonus ako keby do nej narazila.

8.6.7.4 Kolízia náboja s tunelom

Pri tejto kolízii dôjde iba k odstráneniu náboja zo scény, pretože nie je potrebné ho ponechávať v scéne aj keď sa nachádza mimo objektu tunela.

9 Záver

Pri vytváraní tejto bakalárskej práce som sa stretol s väčšími aj s menšími problémami. Prvým väčším problémom, ktorý stojí za spomenutie bolo určite to, že toto bolo pre mňa prvé stretnutie s technológiou Java Monkey Engine a na moje veľké prekvapenie dostupnosť študijných materiálov, či odborných náučných kníh venovaných sa tejto hernej knižnici, je naozaj nízka. Jediným dostupným materiálom bola oficiálna stránka jME a jej fórum. Preto študovanie a zoznámenie sa s touto knižnicou bolo celkom náročné. Druhým problémom, ale o to príjemnejším bolo vytváranie 3D modelov v prostredí Autodesk 3D Studio Max, pretože tento grafický program disponuje veľkou škálou študijných textov a ukážkových návodov. Po prekročení týchto počiatočných problémov sa mi podarilo úspešne zvládnuť techniky potrebné pri tvorbe počítačovej hry v Jave a od tohto momentu sa práca na tejto bakalárskej práci stala pre mňa veľmi zaujímavou a prínosnou, pretože mi umožnila prehĺbiť si znalosti ako z tvorby počítačových hier v Jave, programovaní samotnom, tvorbe 3D modelov v programe 3D Studio Max, tak aj v tvorbe zvukových efektov v programe FL Studio.

Cieľom tejto práce bolo demonštrovať interakciu medzi jME a importovanými 3D modelmi vytvorením počítačovej hry, čo sa mi úspešne podarilo. Toto prepojenie sa snažím podať v atraktívnej forme pre užívateľa, preto sú do hry implementované všetky dôležité súčasti, ktoré by mala počítačová hra v dnešnej dobe obsahovať. Počnúc úvodným načítaním hry, cez príjemné grafické menu, zvukové a vizuálne efekty až po intuitívne a jednoduché ovládanie. Všetky tieto súčasti bolo možné implementovať vďaka prepracovanej hernej knižnici Java Monkey Engine, ktorá vzhľadom k tomu, že sa jedná o open source knižnicu, ponúka širokú paletu možností pri tvorbe hier v Jave, vďaka čomu sa tieto hry dajú tvoriť pomerne rýchlo, jednoducho a efektívne.

V budúcnosti by sa do hry mohla implementovať umelá inteligencia pre pretekanie sa so súpermi, prípadne z tejto hry vytvoriť sieťovú hru pre viacerých hráčov, doplniť hru o editor tratí, umožniť užívateľovi si vytvoriť vlastnú vesmírnu loď, či rozšíriť paletu bonusov a špeciálnych efektov.

Literatúra

[1] Davison, A.: Programování dokonalých her v Javě. Brno: Computer Press, 2006, ISBN 80-7226-944-5, 904 s.

[2] User's guide. [online], [cit. 2010-04-01].
URL <http://www.jmonkeyengine.com/wiki/doku.php?id=user_s_guide>

[3] About jME. [online], [cit. 2010-04-01].
URL <http://www.jmonkeyengine.com/wiki/doku.php?id=about_jme>

[4] OpenGL Utility Library. [online], [cit. 2010-04-01].
URL <http://en.wikipedia.org/wiki/OpenGL_Utility_Library>

[5] LWJGL. [online], [cit. 2010-04-01].
URL <http://www.lwjgl.org/about.php>

[6] Architecture overview. [online], [cit. 2010-04-01].
URL <http://www.jmonkeyengine.com/wiki/doku.php/jme_architecture_overview>

[7] Scene graph. [online], [cit. 2010-04-01].
URL <http://www.jmonkeyengine.com/wiki/doku.php/what_is_a_scene_graph>

[8] Java Virtual Machine. [online], [cit. 2010-04-01].
URL <http://en.wikipedia.org/wiki/Java_Virtual_Machine>

[9] J2ME. [online], [cit. 2010-04-01].
URL <http://sk.wikipedia.org/wiki/J2ME>

[10] Introduction to models. [online], [cit. 2010-04-01].
URL <http://www.jmonkeyengine.com/wiki/doku.php/introduction_to_models>

[11] ASE File Format. [online], [cit. 2010-04-01].
URL <http://wiki.beyondunreal.com/Legacy:ASE_File_Format >

[12] Collada. [online], [cit. 2010-04-01].
URL <http://www.khronos.org/collada>

[13] Game types. [online], [cit. 2010-04-01].
URL <http://www.jmonkeyengine.com/wiki/doku.php/game_types>

[14] FengGUI. [online], [cit. 2010-04-01].
URL <http://www.fenggui.org/doku.php >

[15] Free 3D models. [online], [cit. 2010-04-01].
URL <http://artist-3d.com/free_3d_models>

[16] Definition of Quaternion. [online], [cit. 2010-04-01].
URL <http://www.jmonkeyengine.com/wiki/doku.php/quaternion?s[]=rotation>

[17] Introduction to 3D Studio Max. [online], [cit. 2010-04-01].
URL <http://www.ogle.com/Classes/IVC/Max_Level_1.pdf>

[18] Model loading. [online], [cit. 2010-04-01].
URL <http://www.jmonkeyengine.com/wiki/doku.php/model_loading>

Zoznam príloh

Príloha 1. DVD so zdrojovými textami, programovou dokumentáciou a spustiteľnou aplikáciou.